MFT 3.4.0.0 Connectors Developer Guide

MFT 3.4.0.0 Connectors Developer Guide

Publication date 2019-11-05

EquiSign
76 route de la Demi-Lune
Les Collines de l'arche
92057 Paris La Défense Cedex-France
France
www.equisign.fr

Copyright © 2019 EquiSign. All Rights Reserved.

This product, including its related documentation, is protected by copyright and may be protected by patent.

Restricted Rights. This product, including its associated documentation, is intended to be used exclusively by holders of valid EquiSign licenses for the products documented herein. No part of this document may be reproduced or transmitted, in any form or by any means, without the prior written consent of EquiSign.

Limited Liability. While the utmost precaution has been taken in the preparation of this documentation, EquiSign assumes no responsibility for errors or omissions in this documentation. Information in this document is subject to change without notice and does not represent a guarantee on the part of EquiSign. The documentation is provided "as is" without warranty of merchantability or fitness for a particular purpose. Furthermore, EquiSign does not warrant, guarantee, or make any representations regarding the use, or the results of the use, of the documentation in terms of correctness, accuracy, reliability, or otherwise.

Trademarks and Trade Name. EquiSign® is a registered trademark of EquiSign SAS in the United States and other countries. EquiSign is a trade name of EquiSign SAS in the United States and other countries.

All other brand or product names referred to in this document are registered trademarks, trademarks, service marks, or trade names of their respective owners.

Third-party Reference Documentation. The information contained in this document is not yet part of the standard EquiSign documentation set or is intended to be a reference guide for integrating EquiSign products with third-party software. As such, it is not officially supported or maintained by EquiSign and will not necessarily be updated on the same release schedule as officially supported EquiSign products and their associated documentation.

Pre-release Documentation. The information contained in this document is not yet part of the standard EquiSign documentation set or is intended to be a reference guide for integrating EquiSign products with third-party software. As such, it is not officially supported or maintained by EquiSign and will not necessarily be updated on the same release schedule as officially supported EquiSign products and their associated documentation.

Supplemental Documentation. The information contained in this document is not yet part of the standard EquiSign documentation set or is intended to be a reference guide for integrating EquiSign products with third-party software. As such, it is not officially supported or maintained by EquiSign and will not necessarily be updated on the same release schedule as officially supported EquiSign products and their associated documentation.

Developer Documentation. The information contained in this document was written for developers by developers as a reference tool. It will not necessarily be updated on the same release schedule as EquiSign products and their associated documentation.

Table of Contents

Preface vi
1. Resources vi
1.1. Contact Customer Service vi
1.2. Contact Professional Services vi
1.3. Provide Documentation Feedback vi
2. Document Conventions vi
1. How to Use the Managed File Transfer Connectors 1
1.1. Introduction 1
1.2. Included Reference Documentation 1
1.3. General Design 2
1.3.1. SOAP and REST interfaces 2
1.3.2. Data Marshaling 2
1.3.3. SOAP Method Calling Conventions 3
1.3.4. REST Method Calling Conventions 3
1.3.5. SOAP File Uploads 4
1.3.6. REST File Uploads 4
1.4. Identification 5
1.4.1. Authentication 5
1.4.2. Impersonation 6
1.4.3. Authentication through SOAP Headers 6
1.4.4. Authentication through HTTP Headers 6
1.4.5. Using a Session 6
1.4.6. Header Examples 7
1.5. Error Handling 8
1.5.1. Authentication Errors 8
1.5.2. Other Errors 8
1.5.3. Connector Error Codes 8
1.5.4. SOAP Faults 9
1.5.5. JSON Errors 9
1.5.6. SOAP Fault Examples 10
1.5.7. REST Error Examples 11
1.6. Data Types Handling 11
1.6.1. Date 11
1.6.2. Boolean 12
1.7. Troubleshooting 12
1.7.1. SOAP Connector 12
1.7.2. REST Connector 13
2. Managed File Transfer Admin SOAP/REST Connector Documentation 14
2.1. Compatibility Note 14
2.1.1. API Changes (2.0 to 2.6) 14
2.1.2. API Changes (1.0 to 2.0) 15
2.2. Network Connection 15
2.2.1. SOAP Connector 15
2.2.2. REST Connector 15
2.3. List of Connector Operations 16
2.4. Authentication and Access Control 16
2.5. Detailed Documentation 17
2.5.1. API Description 17
2.5.2. API Methods 20
3. Managed File Transfer Rights SOAP/REST Connector Documentation 37
3.1. Compatibility Note 37
3.1.1. API Changes (1.0 to 1.1) 37
3.2. Network Connection 37
3.2.1. SOAP Connector 37
3.2.2. REST Connector 38

3.3. List of Connector Operations 38
3.4. Authentication and Access Control 38
3.5. Detailed Documentation 39
3.5.1. API Description 39
3.5.2. API Methods 39
4. Managed File Transfer File SOAP/REST Connector Documentation 45
4.1. Compatibility Note 45
4.1.1. API Changes (2.0 to 2.6) 45
4.2. Network Connection 46
4.2.1. SOAP Connector 46
4.2.2. REST Connector 46
4.3. List of Connector Operations 46
4.4. Authentication and Access Control 47
4.5. Detailed Documentation 47
4.5.1. API Description 47
4.5.2. API Methods 52
4.5.3. File Downloads 64
4.5.4. File Uploads 64
A. Schema documentation for otmessage-1.3.xsd 68
A.1. Namespace: "http://www.opentrust.com/OpenTrust/XML/Message/1.3" 68
A.1.1. Schemas 68
A.1.2. Elements 68
A.1.3. Complex Types 72
A.2. Namespace: "" 73
A.2.1. Attributes 73
B. Schema documentation for otmessage-1.4.xsd 74
B.1. Namespace: "http://www.opentrust.com/OpenTrust/XML/Message/1.4" 74
B.1.1. Schema(s) 74
B.1.2. Element(s) 74
B.1.3. Complex Type(s) 79
B.2. Namespace: "" 80
B.2.1. Attribute(s) 80

List of Tables

2.1. Access Rights for SOAP Operations 17

Preface

The following sections contain preface information:

- Section 1, “Resources”
- Section 2, “Document Conventions”

1. Resources

Please use the information provided to contact the appropriate EquiSign department or representative.

1.1. Contact Customer Service

Support Web Site, including the Support Download Site	`https://support.equisign.fr/` (Login requires a username and password)
Email	`<support@equisign.fr>`

1.2. Contact Professional Services

Email	`<support@equisign.fr>`

1.3. Provide Documentation Feedback

As part of an ongoing process to create documentation that is easy to understand and use, as well as relevant to audience roles as administrator users, we welcome feedback about this guide. Please email any comments or suggestions to: `<support@equisign.fr>`

2. Document Conventions

EquiSign documentation uses typographical conventions with specific meanings. These conventions are described in the following table.

Convention	How It Is Used
bold	Indicates the most important part of a step in step-based instructions. Example: Click the OK button.
italic	Indicates a reference to another document or guide. Example: See the Release Notes. Indicates the name of an access right. Example: The unlock right allows an administrator to help an end user unlock a smart card.

Convention	How It Is Used
`monospaced font`	Indicates a file name, directory name or path, code examples and elements, application output, and user-entered text. Example: Save the file in the `/webserver` directory.
`italicized monospaced font`	Indicates an environment-specific or implementation-specific variable. Example: Save the file in the *`root_directory/`*`webserver` directory.
Important:	Contains important information that must be paid attention to. Failure to do so may have a negative impact on the application.
Note:	Contains valuable supplementary information.
Tip:	Contains helpful information that may be useful, for example, a shortcut or another way of performing a task.

Chapter 1. How to Use the Managed File Transfer Connectors

1.1. Introduction

This document is a technical description for the Managed File Transfer Connectors: the Admin SOAP/REST Connector, the Rights SOAP Connector and the File SOAP/REST Connector. It is assumed that the reader is familiar with the following concepts and technologies : XML, XML Schema, SOAP, HTTP, WSDL. Moreover, the reader must fully understand the concepts related to MFT management, such as domains and policies.

The Connectors are an Application Programming Interface (API), which allows external applications to interact with MFT. It may be used to integrate the product with such applications as:

- Identity & Access Management (IAM) platforms:
  - Provisioning of users. Including tasks such as creation, update, deactivation, etc.
  - Role-based Access Control management of MFT by the IAM platform using the SOAP Rights Connector.
- Workflow engines:
  - Sending of documents to workflow actors.

The File SOAP/REST Connector can be accessed using SOAP or REST interfaces. Although the API specifications differ when it comes to authentication and error handling, both interfaces share most of the same requirements.

1.2. Included Reference Documentation

Developers are strongly advised to read the following documents in order to understand how XML messages that are sent or received by the Managed File Transfer SOAP Connectors are formatted. These documents can be found in the same Managed File Transfer Connectors .zip file in which this guide is located.

`otmft-admin-connector-1.1.wsdl`	WSDL (Web Service Definition Language) definition of the Managed File Transfer Admin SOAP/REST Connector API, which can be used to automatically generate stubs to access the Admin SOAP/REST Connector in any programming language.
`otc-rights-connector-1.1.wsdl`	WSDL (Web Service Definition Language) definition of the Managed File Transfer Rights SOAP/REST Connector API, which can be used to automatically generate stubs to access the Rights SOAP Connector in any programming language.
`otmft-file-connector-1.1.wsdl`	WSDL (Web Service Definition Language) definition of the Managed File Transfer File SOAP/REST Connector API, which can be used to automatically generate stubs to access the File SOAP/REST Connector in any programming language.

`otmessage-1.3.xsd`	XML schema of the MFT Message data type used by the prior Managed File Transfer Connectors versions. See Appendix A, Schema documentation for otmessage-1.3.xsd for additional information.
`otmessage-1.4.xsd`	XML schema of the MFT Message data type used by the Managed File Transfer Connectors. See Appendix B, Schema documentation for otmessage-1.4.xsd for additional information.

1.3. General Design

1.3.1. SOAP and REST interfaces

The MFT Connectors may be accessed using either a SOAP or REST interface. Although the REST interface is the preferred way to access the connectors, the SOAP interface provides a technically equivalent method.

The MFT Connectors API does not change depending on the interface, although method call conventions are a bit more complex when using the REST interface for some operations. Please see Section 1.3.4, “REST Method Calling Conventions” for more information.

1.3.2. Data Marshaling

The connectors expose a set of methods (or operations, in SOAP terminology), which all use the same type of input and output message: MFT Message.

An MFT Message has the following characteristics :

- it is structured: it can contain arrays (lists), hash tables (also know as associative arrays or dictionaries), or simple values
- it is not typed: the semantic of the message is not encoded in the message itself; the element names and attributes of the message do not carry any semantic

As a result, the meaning of the message is purely contextual to the operation, and it is the responsibility of the caller to send the proper data to the Connector.

1.3.2.1. SOAP context

When using the SOAP interface, the MFT Message is represented in XML using a documented data type. The name space of this data type is "http://www.opentrust.com/OpenTrust/XML/Message/1.4", and it is defined in the XML Schema named `otmessage-1.4.xsd`.

1.3.2.2. REST context

When using the REST interface, the request data is sent using standard HTTP form data (as per HTTP 1.1) through POST and/or GET requests.

The method arguments must be marshalled in JSON format as a direct rendering of the Hash/Array/Value structure. These arguments are then passed in the HTTP request. See Section 1.3.4, “REST Method Calling Conventions” for more information.

1.3.3. SOAP Method Calling Conventions

The arguments of an operation are wrapped into an MFT Message, which can contain either a single Value, Array, or HashTable.

The MFT Connectors always require that the root type of the MFT Message be an array whose elements are the arguments of the method.

Some methods expect optional arguments, and only one mandatory argument. In this case, even though there may be only one argument to a call to this method, it must still be wrapped into an Array.

1.3.4. REST Method Calling Conventions

The access URI for each operation is composed using the base URI for the REST Connector, to which is appended the method's name (i.e: the `getBogusInformation` REST method URI for a Bogus connector would be `http://app.example.com/app-prefix/connectors/REST/Bogus/getBogusInformation`). For more information on the URI of each REST connector, see the connector's specific documentation. Unless otherwise noted, the preferred method to access the REST connectors is POST. No parameters should be added in the query string, as those would be available in the Audit Logs.

When calling the Connectors REST Interface, the response respects the following conventions:

- When a call is successfull, the HTTP Code 200 is set
- The response encoding is UTF-8
- The response content type is application/json when JSON content is returned, and text/plain when simple text is returned. Note: the server may be configured to set a custom content type when returning JSON data.

When sending the method arguments in JSON form, two strategies may be employed. These are described in the following subchapters.

1.3.4.1. Single args HTTP parameter

The input array is serialized to a single JSON array and set as the value of an `args` HTTP parameter.

For example, the `getBogusInformation` method may be called using the following POST:

```
$ curl -H "X-Otc-Auth-Uid: YWRtaW4ta3dw" \
  -H "X-Otc-Auth-Password: T3BlblRydXN0" \
  --data-urlencode "[ { "bogus_ref":"A23B7" } ]" \
  -v  http://app.opentrust.com/app-prefix/connectors/REST/getBogusInformation
POST /app-prefix/connectors/REST/Bogus/getBogusInformation HTTP/1.1
User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) ...
Host: localhost:8888
Accept: */*
X-Otc-Auth-Uid: YWRtaW4ta3dw
X-Otc-Auth-Password: T3BlblRydXN0
Content-Type: application/x-www-form-urlencoded
Content-Length: 32
args=%5B%20%7B%20%22bogus_ref%22%3A%22A23B7%22%20%7D%20%5B

Non URL encoded JSON value of args:
[ { "bogus_ref":"A23B7" } ]
```

1.3.4.2. application/json payload

The input array is serialized to a single JSON array and sent as the POST body using the `application/json` content type. This method is the preferred way to call the REST connector.

For example, the `getBogusInformation` method may be called using the following curl command:

```
$ curl -H "X-Otc-Auth-Uid: YWRtaW4ta3dw" \
  -H "X-Otc-Auth-Password: T3BlblRydXN0" \
  -H "Content-Type: application/json" \
  -d "[ { "bogus_ref":"A23B7" } ]" \
  -v  http://app.opentrust.com/app-prefix/connectors/REST/getBogusInformation

POST /app-prefix/connectors/REST/Bogus/getBogusInformation HTTP/1.1
User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3
Host: localhost:8888
Accept: */*
X-Otc-Auth-Uid: YWRtaW4ta3dw
X-Otc-Auth-Password: T3BlblRydXN0
Content-Type: application/json
Content-Length: 32
[ { "bogus_ref":"A23B7" } ]
```

Note that the authentication parameters are passed using the HTTP headers methods (see Section 1.4.4, “Authentication through HTTP Headers”), and the content-type is correctly set to `application/json`.

Important

Also it may work, passing the authentication arguments as HTTP parameters and using a multipart MIME request to pass the JSON payload as a separate "application/json" part is NOT supported.

1.3.5. SOAP File Uploads

When a binary payload must be sent in a SOAP call, the binary value must be wrapped in a single BinaryValue element, as defined in the MFT Message 1.4 schema.

The standard Message Transmission Optimization Mechanism (MTOM) is authorized and may be used in such cases.

1.3.6. REST File Uploads

When a binary payload must be sent in a REST call, and unless otherwise noted, the following options are available:

Inline Base64 data	The data is encoded in base64 and directly included in the JSON structure. This may be used for small payloads (less than 100KB).
Reference multipart part	The HTTP call is a MIME Multipart POST, with the JSON request and binary payloads dispatched in different parts. In this case, the file payload must be referenced from the JSON structure using the `cid:partname` syntax.

The following give an example of a JSON structure and using the `curl` command to send the request:

```
[
  {
    subject : "My data",
    files : [
      {
        filename : "Plan.pdf",
```

```
        data: "cid:plan"
      }
    ]
  }
]

$ curl -H "X-Otc-Auth-Uid: YWRtaW4ta3dw" -H "X-Otc-Auth-Password: T3BlblRydXN0" \
-F args=@mydata.json -F "plan=@/tmp/Plan.pdf" \
http://myserver/app-prefix/connectors/REST/Bogus/sendBogusData
```

1.4. Identification

When it accesses the MFT Connectors, the client application must be identified and mapped to a specific MFT user that the client application thus impersonates. All operations realized through the connector will then be traced with this user as the Actor. This usually mandates that a user dedicated to connector operations be created and managed separately from standard users.

The MFT user may be identified through user authentication on each connector call, or by using a session id once an initial authentication has been performed.

1.4.1. Authentication

The client application must be authenticated in order use the MFT Connectors. Since MFT provides a configurable and pluggable authentication system, the authentication schemes that are available to the client application will depend on how the MFT platform is configured. These schemes are the sames as those available to the user impersonated by the client application.

The authentication schemes can work at two different levels:

Transport level	This includes authentication schemes such as SSO or SSL Authentication with a client X509 Certificate. SSL Authentication requires that the client certificate be valid and trusted by the MFT platform.
Message level	This includes all authentication schemes that require a user login and credentials to be passed through the Message. In this case, the credential informations are passed in the either in SOAP Headers or in HTTP headers, as described in the next sections.

When user identification and credential information need to be passed through the Message, they must be provided as named attribute that depend on the the configured authentication plugins (schemes). The following keys are those used by default when LDAP authentication plugin or the builtin authentication mechanism are used.

uid , email or ident	Respectively unique identifier, email or generic identifier or the user. These attributes are mutually exclusive and serve in identifying the user in a unambiguous way. The choice of what key to use depends on the configuration of the application. It must be coherent with the input parameter configured in the Login Customization.
password	The user password, in clear text.
domain	The user domain name. This may be required if the identifying key does not permit unambiguous identification of the user, although this is a rare corner case; most often the domain is not required (this should be checked with the team in charge of the platform's configuration).

1.4.2. Impersonation

In certain circumstances, it may be required that a single "technical" user account be configured to access the MFT Connectors and perform operations on behalf of other users. To impersonate another user, the following requirements must be met:

- The authenticated user must possess the Impersonation right
- The user to impersonate must be unambiguously determined
- The authentification scheme of the "technical" user account will override the authentication scheme of the user to impersonate

To identify the user that should impersonated, additional identification credentials must be provided, in a special Acting As context.

If impersonation data is present, then it must be valid, otherwise the call will fail with a `Client.AccessDenied` error.

1.4.3. Authentication through SOAP Headers

When using the MFT SOAP Connectors, user identification and credential information may be present in the Header part of the SOAP envelope. The header contains a `otmsg:Header` element (conforming to the `otmessage-1.4.xsd` schema), with the `soapenv:mustunderstand` attribute set to `"1"` and the `type` attribute set to `"auth"`. The Header element will contain an `otmsg:HashTable` that gives the required information to authenticate the client application.

Impersonation identification credentials are provided in another `otmsg:Header` element with the `type` attribute set to `"acting_as"`.

1.4.4. Authentication through HTTP Headers

When using the MFT REST Connectors or MFT SOAP Connectors, user identification and credential information may be present in the HTTP Headers or the request. To each credential information corresponds an HTTP Header, named using the following convention:

`"X-OTC-Auth-"` + `AttrName`

The header value must be Base64 encoded.

Thus, an authentication scheme that requires the `ident` and `password` attributes will expect the following Headers in the HTTP request:

X-OTC-Auth-Ident: Base64EncodedIdent

X-OTC-Auth-Password: Base64EncodedPassword

Impersonation identification credentials are provided through HTTP Headers named using the following convention:

`"X-OTC-ActingAs-"` + `AttrName`

1.4.5. Using a Session

When a successful authenticated call to the connector is made, a server side session is created and a session id is returned in the response HTTP headers. This session id may be used in

subsequent calls in place of the authentication credentials and for the duration of the session. Once a session a session is expired, a call to the connector using the session id will fail with a `Client.AccessDenied` error.

Making connector calls with a session id has the following advantages:

- the authentication step is bypassed, which speeds up the request;
- no authentication audit log is generated;

This is especially useful when multiple related connector calls are made by the calling application.

The MFT Connectors session management is that of the Java Servlet Specifications (http://download.oracle.com/otndocs/jcp/servlet-2.4-en-spec-oth-JSpec/), using a Cookie named `JSESSIONID`. To use the session in connector calls, one must:

- make a first connector call with authentication credentials;
- extract the session id from the `Set-Cookie` HTTP Header;
- in subsequent calls, set the `Cookie` HTTP Header with a `JSESSIONID=`*`sessionid`* value.

1.4.6. Header Examples

1.4.6.1. LDAP Authentication SOAP Header

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.opentrust.com/OpenTrust/XML/Message/1.4"
  xmlns:ns1="http://www.opentrust.com/OpenTrust/APP/Connector/Admin/SOAP/1.0">
  <ns:Header name="auth" soapenv:mustunderstand="1">
    <ns:HashTable>
      <ns:Item key="uid">
        <ns:Value>jsmith</ns:Value>
      </ns:Item>
      <ns:Item key="password">
        <ns:Value>mycatisbluewithyellowstripes</ns:Value>
      </ns:Item>
      <ns:Item key="domain">
        <ns:Value>catlovers</ns:Value>
      </ns:Item>
    </ns:HashTable>
  </ns:Header>
  <soapenv:Body>

  ...


  </soapenv:Body>
</soapenv:Envelope>
```

1.4.6.2. Session ID HTTP Headers

HTTP Response after a successful authentication:

```
HTTP/1.1 200 OK
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: JSESSIONID=1thbwyba1mu1z;Path=/app-admin
Content-Type: application/json; charset=UTF-8
Content-Length: 59
```

HTTP Request in subsequent calls:

```
POST /app-admin/connectors/REST/Bogus/listBogusStuff HTTP/1.1
User-Agent: curl/7.19.7 (x86_64-pc-linux-gnu) libcurl/7.19.7 OpenSSL/0.9.8k zlib/1.2.3.3 libidn/1.15
Host: app.acme.ltd
```

```
Accept: */*
Cookie: JSESSIONID=1thbwyba1mu1z
```

1.5. Error Handling

1.5.1. Authentication Errors

Authentication errors are handled either at the HTTP level or at the Message level, depending on what authentication scheme is used: an attempt to authenticate with an invalid client certificate will always result in a 403 HTTP error. However, if the SSL negotiation succeeded, the authentication may fail on the Message level if the given credentials (login/password) are invalid, in which case the client will receive either a 403 HTTP error or a specific Connector Error.

When using the MFT REST Connectors, an authentication error will always result in a 403 HTTP error.

1.5.2. Other Errors

When the MFT Connectors encounter an error, then a specific Connector Error message is returned back to the client. The actual format of the message depends on the interface through which the connector is accessed (REST or SOAP), although the error will always contain the same information:

errorCode	An error code. Its prefix may be either Server or Client, depending on the error's origin. The possible values are detailed below.
errorSummary	A short String describing the error encountered (in English).
errorDetails	An MFT Message type HashTable whose attributes depend on the error code.

1.5.3. Connector Error Codes

The `errorCode` field of the returned Connector Error can have the following values:

Client.IncorrectMessage	This error will be returned if: • the message sent was mal-formed • the message sent by the client was syntaxically incorrect, i.e. if it does not comply with the XML schema or JSON syntax • an unknown method was called
Client.WrongParameter	This error will be returned if the message contains unexpected parameters or if one of the required parameters is missing. For example, if the method expects an Array and a HashTable was sent.
Client.IncorrectParameterSyntax	This error will be returned if the message contains a parameter which is semantically incorrect. For example, if a required

	profile name field is missing or if the string "ten" was sent for a parameter which is supposed to be an integer. The ErrorDetail's message contains the parameter names as keys of its inner HashTable, while the values will have the following possible string : 'invalid' if the syntax of the parameter is invalid, 'missing' if the parameter was missing in the request.
Client.AccessDenied	This error will be returned if the authentication of the principal failed or if it is not authorized to execute the operation.
Client.CannotExecuteOperation	This error will be returned if the operation requested by the client could not be executed for some reason, which must be specified in the hash contained by errorDetail as the value corresponding to the `reason` mandatory key. See Client.CannotExecuteOperation SOAP Fault.
Server.InternalError	This error will be returned if Protect & Sign Portal could not process the request for any other reason. The faultstring and faultdetail fields may be used to further qualify the error.

1.5.4. SOAP Faults

When using the Protect & Sign Portal SOAP Connectors, Connector Errors are transformed into SOAP faults. Such SOAP faults will have its following fields set:

faultcode	This is the Connector Error's errorCode. Custom Fault Codes from KWP all belong to the `http://www.opentrust.com/OpenTrust/OTC/Connector/FaultCodes` namespace as recommended by WS-I Basic Profile.
faultstring	This is the Connector Error's errorSummary.
faultdetail	This is the Connector Error's errorDetail, encapsulated into an otmsg:ErrorDetail XML element with attribute `code` set to one of the documented errorCode values. The ErrorDetail may contain itself an otmsg:Message. This message will be an otmsg:HashTable, which represents the content of the Connector Error's errorDetail element. See Client.IncorrectParameterSyntax SOAP Fault.

1.5.5. JSON Errors

When using the MFT REST Connectors Connector Errors are transformed into JSON Maps that are directly returned as a result of the HTTP request. The JSON representation contains three attributes, `errorCode`, `errorSummary` and `errorDetails` that directly represent the Connector Error elements. See Client.IncorrectParameterSyntax REST Error.

In addition, the HTTP Status Code is set to one of the following values, depending on the `errorCode`:

400 (bad request)	When the client request is invalid, contains missing or incorrect parameters (error code with prefix `Client`).
404 (resource not found)	When the resource could not be found.
500 (internal server error)	When an internal server error occurred (error code with prefix `Server`).

To check for an error when using the MFT REST Connectors, the client application should test the HTTP Error Code and the `errorCode` attribute of the returned JSON Map.

1.5.6. SOAP Fault Examples

1.5.6.1. Client.IncorrectParameterSyntax SOAP Fault

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode xmlns:kwpfaults=
       "http://www.opentrust.com/OpenTrust/OTC/Connector/FaultCodes">
       kwpfaults:Client.IncorrectParameterSyntax</faultcode>
      <faultstring>invalid syntax</faultstring>
      <detail>
        <ErrorDetail code="Client.IncorrectParameterSyntax"
         xmlns="http://www.opentrust.com/OpenTrust/XML/Message/1.3">
          <HashTable>
            <Item key="authenticationProfileName">
              <Value>invalid</Value>
            </Item>
            <Item key="userPassword">
              <Value>missing</Value>
            </Item>
          </HashTable>
        </ErrorDetail>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1.5.6.2. Client.CannotExecuteOperation SOAP Fault

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <soap:Fault>
         <faultcode xmlns:kwpfaults=
          "http://www.opentrust.com/OpenTrust/OTC/Connector/FaultCodes">
          kwpfaults:Client.CannotExecuteOperation</faultcode>
         <faultstring>RightsConnector.addGroupToGroup: group cycle detected.</faultstring>
         <detail>
            <ErrorDetail code="Client.CannotExecuteOperation"
             xmlns="http://www.opentrust.com/OpenTrust/XML/Message/1.3">
               <HashTable>
                  <Item key="reason">
                     <Value>E_GROUP_CYCLE_DETECTED</Value>
                  </Item>
               </HashTable>
            </ErrorDetail>
         </detail>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

1.5.6.3. Other SOAP Fault

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Server</faultcode>
      <faultstring>java.lang.NullPointerException</faultstring>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

1.5.7. REST Error Examples

1.5.7.1. Client.IncorrectParameterSyntax REST Error

```
{
    "errorCode": "Client.IncorrectParameterSyntax",
    "errorSummary": "Invalid and/or missing parameter(s)"
    "errorDetails": {
        "id": "invalid"
        "subject": "missing"
    },
}
```

1.5.7.2. Client.CannotExecuteOperation REST Error

```
{
    "errorCode": "Client.CannotExecuteOperation",
    "errorSummary": "Uploaded file not found!"
    "errorDetails":     {
        "details": "File MonPremierFichier.doc could not be found in user upload queue!",
        "reason": "NOT_FOUND"
    },
}
```

1.6. Data Types Handling

1.6.1. Date

> **Important**
>
> The time zone for the date is ALWAYS assumed to be UTC.

Date parameters conform to the following conventions:

input parameters

The string that represents the date data type must use one of the following syntax. The presence of the time part depends on the parameter's semantics.

- 2000-12-25
- 20001225
- 2000-12-25 12:25:35
- 20001225 12:25:35
- 20001225122535Z (YYYYMMDDHHMMSS, the Z is for Zulu and denotes the UTC Timezone)

If the time is omitted in a date-time context, it is assumed that the time is equal to 00:00:00.

The now keyword may be used and will be interpreted as the current date and time.

output parameters	The following syntax is always used: 20001225122535Z

1.6.2. Boolean

Boolean parameters conform to the following conventions:

input parameters	If the parameter is either `1`, `true` or `yes` (case insensitive), then it is considered a `true` boolean value. Otherwise it is considered a `false` boolean value.
output parameters	`1` for a `true` boolean value, `0` for a `false` boolean value.

1.7. Troubleshooting

When experiencing problems with the connector, the following log files should be inspected:

`/opt/opentrust/mft/var/log/*/tech.log`	This is the main technical log file, for both the admin and user machines.
`/opt/opentrust/mft/var/log/httpd_access.log`	This is the HTTP server access log file, it provides logs of all HTTP calls to the server.
`/opt/opentrust/mft/var/log/httpd_error.log`	This is the HTTP server error log file, it provides logs of all HTTP errors that occurred on the server.

In order to help application developers to diagnose and troubleshoot problems with the connector, MFT can also be configured to output diagnostic messages in the logs regarding the operations of the connector.

To configure this debugging feature, go to `Server Management | Setup | Advanced Settings` in the connnector part of the "Base" Tab on the MFT Administration Interface and set the `connector.soap.dump` or `connector.rest.dump` (depending on the connector interface that is used) property to `true`.

> **Important**
>
> This option MUST NOT be turned on in a production system, because this debug information sent to this file may contain sensitive information (like connecting user's credentials and user passwords).

1.7.1. SOAP Connector

With the dump option turned on, the connector will write the following elements to `/opt/opentrust/mft/var/log/tech.log` (at the INFO level):

- Input SOAP messages (body of the input HTTP request)
- Headers of the input HTTP request (in particular, the `SOAPAction` header)
- Output SOAP messages (body of the output HTTP response)
- Deserialized input parameters, as a perl data structure
- Ouput parameters, as a perl data structure

1.7.2. REST Connector

With the dump option turned on, the connector will write the following elements to `/opt/opentrust/mft/var/log/tech.log` (at the INFO level):

- Headers of the input HTTP request
- Deserialized input parameters, as a perl data structure
- Ouput parameters, as a perl data structure

Chapter 2. Managed File Transfer Admin SOAP/REST Connector Documentation

2.1. Compatibility Note

This documentation is for the version 2.6 of the Managed File Transfer Admin SOAP/REST Connector API. However, the current version of MFT (3.4.0.0) may be backward compatible with previous versions of the API. The following table summarizes the different versions of both the API, and Data marshaling messages supported in this version of MFT:

API Version	MFT Version	API WSDL name space	Message name space	Compatibility
2.6	MFT 3.1.X X >=0 MFT 2.6.X X >=0	http://www.opentrust.com/OpenTrust/MFT/Connector/Admin/SOAP/1.1	http://www.opentrust.com/OpenTrust/XML/Message/1.4 (MFT Message)	No
2.0	MFT 2.0.X X >=0 MFT 2.1.X X >=0 MFT 2.5.X X >=0	http://www.opentrust.com/OpenTrust/MFT/Connector/Admin/SOAP/1.0	http://www.opentrust.com/OpenTrust/XML/Message/1.3 (MFT Message)	Yes
1.0	MFT 1.0.X X >=0	http://www.opentrust.com/OpenTrust/MFT/Connector/Admin/SOAP/1.0	http://www.opentrust.com/OpenTrust/XML/Message/1.3 (MFT Message)	No

2.1.1. API Changes (2.0 to 2.6)

This section lists the changes between the versions 2.0 and 2.6 of the Managed File Transfer Admin SOAP/REST Connector that may have an impact on the use of the connector. Changes that do not have a compatibility impact are not listed. The changes are listed in the following table:

Operations	Changes
all	The migration to OTMessage 1.4 triggered a change in the WSDL namespace of the Managed File Transfer Admin SOAP/REST Connector and an API version bump.
createUser, updateUser	The "locale" parameter is now optional: When missing (creation) or set to "null", the user's Domain default language is used instead.

2.1.2. API Changes (1.0 to 2.0)

This section lists the changes between the versions 1.0 and 2.0 of the Managed File Transfer Admin SOAP/REST Connector that may have an impact on the use of the connector. Changes that do not have a compatibility impact are not listed. The changes are listed in the following table:

Operations	Changes
createUser, getUser, updateUser	The User HashTable definition has changed: The "internal_exchange_policy", "external_exchange_policy", "upload_token_exchange_policy" and "project_exchange_policy" attributes have been replaced by a single "exchange_policy" to reflect the change in MFT handling of users' exchange policies.
getDomainProperties	The "user_quota_policy" attribute in the "quota" map has been renamed to "default_quota_policy" for consistency.
getDomainProperties	The "exchange_policies" attribute in the "available" map now directly returns the list of available exchange policies (no more indirections to internal and external).

2.2. Network Connection

2.2.1. SOAP Connector

This connector is only accessible via HTTP/S, and the SOAP address location is the following:

`https://mft.example.com/mft/connectors/OTMFTAdminConnectorSOAPService` to address the current version of the connector

or

`https://mft.example.com/mft/connectors/OTMFTAdminConnectorSOAPService_X_Y` to address the specific X.Y WSDL namespace

Where *`mft.example.com`* is the hostname of the machine where the MFT Administration Webapp is installed.

The *`mft`* prefix refers to the name of the Administration Webapp.

2.2.2. REST Connector

This connector is only accessible via HTTP/S, and the base address location is the following:

`https://mft.example.com/mft/connectors/REST/Admin/`

Where *mft.example.com* is the hostname of the machine where the MFT Admin Webapp is installed.

The *mft* prefix refers to the name of the Administration Webapp.

Considering a function called doStuff, the REST URI for the function will be `https://mft.example.com/mft/connectors/REST/Admin/doStuff`.

2.3. List of Connector Operations

This sections lists the methods exposed by the Managed File Transfer Admin SOAP/REST Connector. The methods arguments and returned objects are described in the next chapter.

createTag	Create a new Tag
createUser	Create a new User
createUsers	Create Users in batch mode
deleteTag	Delete an existing Tag
deleteUser	Delete an existing User
deleteUsers	Delete existing Users in batch mode
getDomainProperties	Get a MFT Domain configuration properties. Notably, this provides the list of profiles available for the domain.
getTag	Get an existing Tag's information
getUser	Get an existing User's information
listDomains	List all configured MFT Domains
searchForUsers	Search for Users using configurable criteria
syncUsers	Synchronize users with a provided, file based datasource
updateTag	Update an existing Tag
updateUser	Update an existing User
updateUsers	Update existing Users in batch mode
version	returns version information about the Managed File Transfer Admin SOAP/REST Connector

2.4. Authentication and Access Control

The SOAP client application is authenticated using the processes described in Section 1.4, “Identification”.

Every operation needs the client application to be authenticated.

In order to call any SOAP operation, the client application needs to be granted rights to access the Managed File Transfer Admin SOAP/REST Connector. The following table details for each operation the right needed to access it. See the relevant section of the Managed File Transfer Server Configuration Guide to learn how to grant those rights to the user impersonated by the SOAP client application.

Table 2.1. Access Rights for SOAP Operations

SOAP Operation	Right Needed	Additional Information
createTag	Config Management (global or loca)	The right must be granted on the tag's Domain, or be global.
createUser & createUsers	User Management	The right must be granted on the user's Domain.
deleteTag	Config Management (global or loca)	The right must be granted on the tag's Domain, or be global.
deleteUser & deleteUsers	User Management	The right must be granted on the user's Domain.
getDomainProperties	Domain Management, Settings Management or any Local Administration Right	The right must be granted on the relevant Domain.
getTag	Config Management (global or local)	The right must be granted on the tag's Domain, or be global.
getUser	User Management	The right must be granted on the user's Domain.
listDomains	Domain Management, Settings Management or any Local Administration Right	Only the authorized Domains are listed.
searchForUsers	User Audit	Only the users belonging to the Domain(s) on which the right is given will be returned as search results.
syncUsers	User Management	Only the users belonging to the Domain(s) on which the right is given may be managed.
updateTag	Config Management (global or local)	The right must be granted on the tag's Domain, or be global.
updateUser & updateUsers	User Management	The right must be granted on the user's Domain.
version	-	No specific right is needed.

2.5. Detailed Documentation

2.5.1. API Description

2.5.1.1. API Description

2.5.1.1.1. Name

com.opentrust.otc.connector.admin.MFTAdminConnector - API for interacting with OpenTrust MFT Admin Connector

2.5.1.1.2. Description

com.opentrust.otc.connector.admin.MFTAdminConnector enables execution of synchronous operations programmatically on the OpenTrust MFT Administration server. It exposes methods such as createUser(), deleteUser(), etc. It can thus be used to manage user provisioning in MFT.

The OpenTrust MFT Admin Connector is a general purpose API that allows an external application to interact with the MFT Administration server.

The main purpose of this class is to be remotely called by a distant client through a dedicated communication layer (for example SOAP). However, it does not implement the communication layer.

Note on the internal identifiers

This API uses several internal identifiers for users and profiles. Although these identifiers are stored as integers, the caller MUST NOT rely on this internal data type. They must be seen as "opaque" ASCII strings (with NO size limit), that should be used as "handlers" to identify the corresponding objects. Such an identifier is guaranteed to be unique among the type of data (i.e. profile, group..) they identify.

2.5.1.1.3. User HashTable

In method calls, a user is described using a `otmsg:HashTable` with the following keys (the mandatory/optional/forbidden aspect of each key depends on the method):

`id` (String - long value)	internal user identifier. This id is managed internally by MFT and should not be set by the caller.
`uid` (String)	unique user identifier. This uid will typically be the user login identifier (such as 'jsmith'). It may also be the user's email. It must be unique on a given domain and cannot be changed after the user is first created.
`email` (String)	user email. The email must be unique on a given domain.
`active` (String - boolean value)	whether the user account is active or not.
`expiration_date` (String - date inputValue)	the expiration date of the user account. If not set on creation the account will never expire. An empty inputValue can be set on update in order to make the user account never expire.
`last_name` (String)	user last name.
`first_name` (String)	user first name.
`domain` (String)	the name of the domain to which the user is attached.
`user_provider` (String)	the name of the user provider to which the user is linked to, if any. The key is not present if the user is not linked to a user provider.
`last_sync_date` (String - timestamp value)	the date of the last synchronization with the linked user provider, if any. The key is not present if the user is not linked to a user provider.
`datasource_uniqueid` (String)	the unique identifier of this user in its user provider's datasource, if any. The key is not present if the user is not linked to a user provider.

`locale` (String)	the preferred locale for the user. This must be a standard locale of the form `langcode` where `langcode` is the language code as defined by ISO-639 (in lowercase) - e.g.: fr, en, es. The country code is not supported in this version (e.g: if fr_FR is used, only the 'fr' part will be taken into account).
`quota_policy` (String)	the name of the quota policy for this user. If not set, the default policy for users of the domain is set instead. This is ignored if user quotas are not enabled on the domain.
`authentication_policy` (String)	the name of the authentication policy for this user. When present it cannot be empty. If not set, the default policy for users of the domain is set instead.
`authentication_policy_alternate` (String - boolean value)	whether the authentication policy is in alternate mode (alternate authentication). If not present, then the alternate mode is `false`.
`authentication_policy_alternate_expiration_date` (String - date value)	the expiration date of the authentication policy alternate mode. Only valid if alternate mode is `true`. This attribute is only returned by the connector, it will be ignored if sent to the connector.
`password` (String)	MFT internal password for the user. This password is never returned by the API but may be set through it. It may not be a requirement for the user account if the authentication policy does not contain the internal authentication scheme.
`exchange_policy` (String)	the name of the exchange policy for this user. If set to an empty string, the user will not be able to send files except through projects of which he is a member. If not set, the default policy for users of the domain is set instead.
`tags` (Array of Strings)	tags applied to the user. If not set, no tags are applied.
`connector_upload_dir` (String)	the connector upload dir of the user. This should be set only if the user account is supposed to be using the File Connector (in offline mode). If not set or empty, no upload directory is applied, and the user cannot use the File Connector.
`custom_attrs` (HashTable)	custom attributes for the user. The HashTable is a simple `String => String` association map that defines a subset of named attributes in {custom1, curtom2, custom3, curtom4} and values for the user.

Tip: the policies available on a domain can be retrieved using the `com.opentrust.mft.connector.admin.MFTAdminConnector.getDomainProperties` method.

2.5.1.1.4. Tag HashTable

In method calls, a tag is described using a `otmsg:HashTable` with the following keys (the mandatory/optional/forbidden aspect of each key depends on the method):

`id` (String - long value)	internal tag identifier
`name` (String)	tag name
`domain` (String)	tag domain name, if it is local to a domain, or an empty string if is a global tag.

`type` (String)	type of tag to be created. For now, only 'user' type tags are supported. If present, must contain `user`.
`description` (String)	tag description.
`is_used` (String - boolean value)	indicates if the tag is affected to at least a user. This attribute is only returned by the connector, it will be ignored if sent to the connector.

2.5.1.1.5. Profile Scope

When referencing profile-type elements (such as Authentication Policies, User Quota Policies, etc) that can have either a Global or Local scope, the following syntax is expected:

- Global Scope - `PROFILE NAME`
- Local Scope - `DOMAIN NAME/PROFILE NAME`

Should a profile name contain a "/" character, then this character must be escaped as "//".

For example, an authentication policy named "LDAP Company B" local to domain "CompanyB" will be referenced as `CompanyB/LDAP Company B`, while a global authentication policy named "LDAP Global" will simply be referenced as `LDAP Global`. A global Tag named "R&D / Q&A" will be referenced as `R&D // Q&A`.

This is applicable to all the operations of the connector where a profile is referenced using a single field.

2.5.2. API Methods

2.5.2.1. createTag(Map<String, Object>)

Creates a new tag.

Expected Argument: Array

request (HashTable)

The only argument. Contains a tag definition as defined in Section 2.5.1.1.4, “Tag HashTable”. The `id` key is forbidden. The following keys are mandatory:

`name`

`domain` (if the tag is global, this attribute must be empty)

Return Value: HashTable

Returns a hash containing the tag information as saved by MFT.

Exceptions:

`Client.AccessDenied`	if the client has no User Administration right.
`Client.WrongParameterException`	if some parameter is missing.
`Client.IncorrectParameterSyntax`	If the syntax of one of the arguments is incorrect or if an argument is missing.

`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

2.5.2.2. createUser(Map<String, Object>)

Creates a new user.

Expected Argument: Array

request (HashTable)

The only argument. Contains a user definition as defined in Section 2.5.1.1.3, “User HashTable”. The `id` key is forbidden. The following keys are mandatory:

`uid`

`email`

`first_name`

`last_name`

`domain`

`active`

Depending on the Authentication Policy of the user, the `password` may or may not be mandatory. When a user password is mandatory, one of the following key must be provided:

`password`	the user password, provided explicitely by the caller
`password_random` (String - boolean value)	must be set to `1`, in which case a password is automatically generated for the user (the password will conform to the attached Password Policy if there is one)

When any of the following optional keys is missing, then the corresponding default value from the user's Domain will be set instead:

`locale`

`authentication_policy` (if present, cannot be empty)

`exchange_policy` (if present, may be empty)

`locale` (if present, may be empty)

`quota_policy` (if present, cannot be empty)

Return Value: HashTable

Returns a hash containing the user information as saved by MFT.

Exceptions:

`Client.AccessDenied`	if the client has no User Administration right.
`Client.WrongParameterException`	if some parameter is missing.

`Client.IncorrectParameterSyntax`	If the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	If the operation could not be completed for some reason.

2.5.2.3. createUsers(Object[], Map<String, Object>)

Creates new users.

Expected Argument: Array

users (Array)

Contains an array of user definitions as defined in Section 2.5.1.1.3, “User HashTable”, with the same conventions as in `com.opentrust.mft.connector.admin.MFTAdminConnector.createUser`.

control (HashTable)

Contains some attributes that are used to control how the batch processing must be handled:

`fail_on_error` (String - boolean)	indicates whether creation errors should stop the processing or not. Defaults to `true`.
`return_user_data` (String - boolean)	indicates whether a full user map should be returned for each processed entry, or a simple success/failure information. For performance and if the user data is not used by the caller, this should be set to `false`. Defaults to `true`.

Return Value: HashTable

Returns a hash with the following structure:

```
{
  nb_success => "2",
  errors => [
              "0", "3"
            ]
  results => [
               {
                 error_code     => "Client.AccessDenied",
                 error_msg      => "Principal does not have right to create\
                                    a user on the Domain 'MyDomain'",
               },
               {
                 id             => 45,
                 uid            => "jsmith",
                 email          => "jsmith@acme.ltd",
                 domain         => "ACME",
                 ...
               },
               {
                 id             => 46,
                 uid            => "asmith",
                 email          => "asmith@acme.ltd",
                 ...
               },
               {
                 error_code     => "Client.IncorrectParameterSyntax",
                 error_summary  => "Unknown domain.",
                 error_details  => "Domain 'toto' is unknown.",
               },
             ]
}
```

The returned hash contains the following attributes:

`nb_success` (String - long value)	The number of users that were successfully created.

`errors` (Array of Strings - long values)	When present, gives the indexes of users that could not be created. When not present, this means that all user creations were successful
`results` (Array of Hashes)	This is an array giving, for each user creation request, a result entry. This array may be shorter than the requested number of user creations if the `fail_on_error` control attribute was set, in which case the last entry is an error. Each entry contains a user definition as defined in Section 2.5.1.1.3, "User HashTable", or just the newly created user id, if the `return_user_data` control attribute was unset.

Exceptions:

`Client.AccessDenied`	if the client has no User Administration right.
`Client.WrongParameterException`	if some parameter is missing.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

2.5.2.4. deleteTag(Map<String, Object>)

Deletes an existing tag.

Expected Argument: Array

request (HashTable)

The only argument. Contains the elements that identify the tag in a non ambiguous manner. The HashTable argument must contain the following keys:

`name`, `domain` (String, mandatory)	the tag name and domain that identify the tag that should be updated. It is mandatory to provide the domain attribute (empty string for a global tag).

Return Value: String

This method returns "1" on success or throws an error (see below).

Exceptions:

`Client.AccessDenied`	if the client has no User Administration right.
`Client.WrongParameter`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason (namely the tag was not found).

2.5.2.5. deleteUser(Map<String, Object>)

Deletes an existing user.

Expected Argument: Array

request (HashTable)

The only argument. Contains the elements that identify the user in a non ambiguous manner. The HashTable argument must contain the following keys:

`uid`, `email` OR `id` (String, mandatory)	the user unique identifier, email or internal identifier. The three elements are mutually exclusive.
`domain` (String, optional)	domain of the user (this is needed if the uid or email does not unambiguously identify the user). This key value is ignored if the id is specified. Warning: identifying a user by her uid/email without giving her domain will cause exceptions to be thrown if two users in different domains have the same uid/email.

Return Value: String

This method returns "1" on success or throws an error (see below).

Exceptions:

`Client.AccessDenied`	if the client has no User Administration right.
`Client.WrongParameter`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason (namely user not found or already deleted).

2.5.2.6. deleteUsers(Object[], Map<String, Object>)

Deletes existing users.

Expected Argument: Array

users (Array)

Contains an array of user identification maps with the same conventions as in `com.opentrust.mft.connector.admin.MFTAdminConnector.deleteUser`.

control (HashTable)

Contains some attributes that are used to control how the batch processing must be handled:

`fail_on_error` (String - boolean)	indicates whether creation errors should stop the processing or not. Defaults to `true`.

Return Value: HashTable

Returns a hash with with the same conventions as `com.opentrust.mft.connector.admin.MFTAdminConnector.createUsers` when the `return_user_data` control attribute is unset:

```
{
  nb_success => "2",
  errors => [
              "0", "3"
            ]
  results => [
               {
                 error_code     => "Client.AccessDenied",
                 error_msg      => "Principal does not have right to create\
                                    a user on the Domain 'MyDomain'",
               },
               {
                 id => 45,
```

```
              },
              {
                id => 46
              },
              {
                error_code     => "Client.IncorrectParameterSyntax",
                error_summary  => "Unknown domain.",
                error_details  => "Domain 'toto' is unknown.",
              },
            ]
}
```

Exceptions:

`Client.AccessDenied`	if the client has no User Administration right.
`Client.WrongParameterException`	if some parameter is missing.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

2.5.2.7. getDomainProperties(String)

Get the configuration properties of a MFT domain.

Expected Argument: Array

domain (String)

The only argument. This is the name of the domain, as returned by the `com.opentrust.mft.connector.admin.MFTAdminConnector.listDomains` method.

Return Value: HashTable

Returns the properties of the MFT Domain configuration as a HashTable with the following structure:

```
 {
  id => "4",
  name => "domain1",
  general => {
               short_appname     => "OpenTrust MFT",
               full_appname      => "OpenTrust Managed File Transfer",
               contact_email     => "mft-contact@somedomain.ltd",
               default_language  => "en",
               motd              => "standard_welcome_motd"
               default_authentication_policy  => "X509 Auth Policy",
               default_exchange_policy        => "Full Rights Exchange Policy"\
,
             },
  quota => {
             has_domain_quota => "1",
             domain_quota => "2000",
             has_user_quota => "1",
             default_quota_policy => "15Mo_quota_pol"
           },
  advanced => {
                 access_url     => "https://mft.opentrust.com/zephyr",
                 sending_policy => "domain1 sending policy",
                 max_user_count => 2000,
                 current_user_count => 100
               },
  available => {
                 authentication_policies => [
                                  "certificate_auth_pol",
                                  "partner_auth_pol"
                                       ],
                 quota_policies => [
                                  "15Mo_quota_pol",
                                  "50Mo_quota_pol",
                                  "100Mo_quota_pol",
                                  "unlimited_quota_pol"
```

```
                                ],
                sending_policies => [
                                    "common",
                                    "confidential"
                                ],
                exchange_policies => {
                                    internal => [
                                            "local_restr_pol",
                                            "domain_restr_pol",
                                            "sales_restr_pol"
                                                ],
                                    external => [
                                            "nolimit_restr_pol",
                                            "partner_restr_pol"
                                                ],
                                },
                 tags => {
                           user => [
                                    "opentrust",
                                    "active_dir_sync",
                                    "vip"
                                   ],
                         },
             },
}
```

The grouping keys are the following:

`general`	general information about the domain
`quota`	quota configuration of the domain
`advanced`	advanced / technical configuration of the domain
`available`	configuration elements (policies...) available on the domain (to the domain and to users of the domain).

The HashTable keys are the following:

`id` (String)	the domain's internal identifier. It can be used to reference a domain.
`name` (String)	the domain's name. It can be used to reference a domain.
`short_appname` (String)	the short application name.
`full_appname` (String)	the full application name.
`contact_email` (String)	the email of the identified contact for the domain
`default_language` (String)	the default language used for notifications
`motd` (String)	the name of the message of the day configured for the domain.
`default_authentication_policy` (String)	the default user authentication policy name for new users.
`default_exchange_policy` (String)	the default user exchange policy name for new users.
`has_domainquota` (Boolean)	whether the domain has domain quota activated
`domain_quota` (String)	the domain quota (if available), in megabytes
`has_userquota` (Boolean)	whether the domain has user quota activated
`default_quota_policy` (String)	the default user quota policy for new users (or users with no quota if they were created prior to enabling user quotas).

`access_url` (String)	the access url for this particular domain
`sending_policy` (String)	the name of the sending policy configured for simple messages
`max_user_count` (String)	the maximum number of user accounts allowed on the domain (if equal to 0, then there is no limitation)
`current_user_count` (String)	current number of user accounts existing on the domain
`authentication_policies` (Array of Strings)	the names of the authentication policies that are available to users of the domain.
`quota_policies` (Array of Strings)	the names of the quota policies that are available to users of the domain.
`sending_policies` (Array of Strings)	the names of the exchange policies that are available to users of the domain.
`exchange_policies` (HashTable)	the names of the exchange policies that are available to users of the domain, accessible through the grouping keys internal and external.
`tags` (HashTable)	the names of the tags that are available to users of the domain, accessible through the grouping key user. This indirection is there as a provision for future extensions of MFT tags.

Note: when an attribute is not set, its value is an empty string (unless otherwise noted).

Exceptions:

`Client.AccessDenied`	if the client has no right to access the domain.
`Client.WrongParameterExcept`	if the domain parameter is missing.
`Client.CannotExecuteOperati`	with reason NOT_FOUND, if the named domain could not be found.

2.5.2.8. getTag(Map<String, Object>)

Returns information about a tag.

Expected Argument: Array

request (HashTable)

The only argument. Contains the elements that identify the tag in a non ambiguous manner. The HashTable argument must contain the following keys:

`name`, `domain` (String, mandatory)	the tag name and domain that identify the tag that should be updated. It is mandatory to provide the domain attribute (empty string for a global tag).

Return Value: HashTable

Returns a hash containing the tag information as defined in Section 2.5.1.1.4, “Tag HashTable”.

Exceptions:

`Client.AccessDenied`	if the client has no User Administration right.

`Client.WrongParameter`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

2.5.2.9. getUser(Map<String, Object>)

Returns information about a user.

Expected Argument: Array

request (HashTable)

The only argument. Contains the elements that identify the user in a non ambiguous manner. The HashTable argument must contain the following keys:

`uid`, `email` OR `id` (String, mandatory)	the user unique identifier, email or internal identifier. The three elements are mutually exclusive.
`domain` (String, optional)	domain of the user (this is needed if the uid or email does not unambiguously identify the user).

Return Value: HashTable

Returns a hash containing the user information as defined in Section 2.5.1.1.3, "User HashTable".

Exceptions:

`Client.AccessDenied`	if the client has no User Administration right.
`Client.WrongParameter`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

2.5.2.10. listDomains()

List the configured domains in the target MFT platform that are accessible (given the client rights).

Expected Argument: none

Return Value: Array of Strings

Returns the list of available MFT Domains for the connecting user as an array containing the domain names as strings. These names can be used directly in the AdminConnectorImpl#getDomainProperties(String) method to access a domain configuration definition.

Exceptions:

`Client.AccessDenied`	if the client has no Configuration right.

2.5.2.11. searchForUsers(Map<String, Object>)

Search MFT users.

Call Example:

```
{
  filter => {
    "uid"                   => "*smith",
    "tags"                  => [ "opentrust", "sync" ],
    "!domain"               => "sales",
    "authentication_policy"  => [ "auth1", "auth2" ],
  },
  fields       => [ "uid", "email", "domain", ],
  limit        => 10,
  count        => 0,
}
```

Expected Argument: Array

request (HashTable)

The only argument. Contains the elements that define the search parameters and expected returned fields. The HashTable must contain the following keys:

`filter` (HashTable, mandatory)

The `filter` argument contains every criterion that the searched users must match. The available criteria are given below. The "*" wildcard character may be used to broaden the scope of the search for text criteria. The search criteria are "AND"-ed together, that means that the returned user will match EVERY given search criterion.

The inputValue of the search criterion may be an array (like for the tag and authentication_policy search criteria above). In this case, the returned users will have to match one of the criterion value in the array. When the user's attribute has multiple values, only one of the values needs to match the criterion: in the example, a user with the tags ("windows", "accountant") will not match, while one with the tags ("windows", "sync") will.

A ! character may be added in front of any criterion name to perform a negative search. In the above example `"!domain" => "sales"` means that the search operation will return every user whose domain is not "sales".

Available criteria:

- uid
- email
- active (boolean, use "1" for true, "0" for false) defaults to **`true`**
- domain
- domain_id
- user_provider
- user_provider_id
- last_name
- first_name
- authentication_policy (no wildcard allowed)

	• authentication_policy_id • authentication_policy_alternatemode • tags (no wildcard allowed) • custom1 • custom2 • custom3 • custom4 • connector_upload_dir
`fields` (Array, optional)	The `fields` argument tells which user attributes should be returned by the search function. If the fields argument is not present, a default list of fields will be returned. Available fields: • uid • email • active • domain • user_provider • last_sync_date • datasource_uniqueid • last_name • first_name • expiration_date • authentication_policy • authentication_policy_alternatemode • tags • custom1 • custom2 • custom3 • custom4 • connector_upload_dir Default fields: uid, email, active, domain, last_name, first_name.
`limit` (String - integer inputValue, optional)	The optional `limit` argument limits the number of returned user entries. This inputValue should be smaller or equal

	than the limit configured on the target MFT platform. If this inputValue is not defined or greater than the configured one, then the latter limit will be used.
`count` (String - boolean inputValue, optional)	The optional `count` argument specify if the result should only be an integer inputValue which represents the number of results.

Return Value: Array

The returned inputValue of this function is an array with an integer in the first element if the argument count is set or an array containing the user entries matching the given search criteria. Each user entry will be a HashTable whose keys are the fields given as arguments (or the default field set).

Exceptions:

`Client.AccessDenied`	if the client has no User Audit right.
`Client.WrongParameterException`	if some parameter is missing.
`Client.IncorrectParameterSyntax`	If the syntax of one of the arguments is incorrect or if an argument is missing.

2.5.2.12. syncUsers(Map<String, Object>)

Synchronize the users in a User Provider realm.

Expected Argument: JSON data referencing the User Provider and representing the users

Important: this operation is only available on the REST connector with strict semantics:

1. the call must be authenticated through HTTP headers or with a JSESSION ID
2. the call must an HTTP POST with the body consisting of a JSON payload
3. the content type must be set to application/json
4. the encoding must be UTF-8

Note that these are the recommended semantics for any call to the REST Connector. However, they are enforced for this particular operation.

The JSON Data must have the following structure:

```
{
  "user_provider" : "File Upload",
  "data" : [
    {
      "uid" : "jsmith",
      "email" : "john.smith0@acme.ltd",
      "first_name" : "John",
      "last_name" : "Smith"
    },
    {
      "uid" : "jdoe",
      "email" : "jane.doe@acme.ltd",
      "first_name" : "Jane",
      "last_name" : "Doe",
      "company" : "Acme, LTD",
      "group" : "Business Development",
      "phone1" : "+33111111111",
      "other1" : "A534VQ324F"
    },
    ...
  ]
}
```

`user_provider` (String)	The qualified name of the targeted User Provider.
`data` (Array)	The exhaustive list of user definitions on this User Provider realm, as a list of hashes. Each hash contains properties named in conformance with the File Upload builtin DataSource. The complete list of mandatory properties depends on the User Provider configuration, please refer to the general documentation of OpenTrust MFT.

Note: the JSON Data must be be syntactically correct and each user definition must be complete before the synchronization process begins (i.e. if there are mandatory fields missing, the synchronization will fail).

Return Value: HashTable

Returns a hash containing a synchronization report, with the following structure:

```
 {
   "status": "PARTIAL",
   "nb_created": "0",
   "nb_updated": "497",
   "nb_removed": "0"
   "processing_time": "27891",
   "errors": [
     {
       "message": "A ValidationException occurred: [Validation error for prope\
rty:
           'Uid': The user ID is already used.][Validation error for proper\
ty:
           'Email': The email address is already used.]",
       "user_index": "13",
       "user_uid": "jsmith11",
       "code": "MAPPING_ERROR"
     },
     {
       "message": "A ValidationException occurred: [Validation error for prope\
rty:
           'Email': The email address is already used.]",
       "user_index": "156",
       "user_uid": "jsmith238bis",
       "code": "MAPPING_ERROR"
     }
   ],
 }
```

`status`	One of COMPLETE (synchronization completed), PARTIAL (synchronization completed with some errors - some users, maybe all, were not synchronized), CANCELLED (synchronization was user cancelled - cannot happen using the connector) or ERROR (the synchronization process never took place, no user was altered).
`nb_created, nb_updated` and `nb_removed`	Respectively, the number of users created, updated and removed during the synchronization.
`processing_time` (time in milliseconds)	The time it took to perform the synchronization.
`errors`	An array of hashes describing the errors that were encountered while performing the synchronization. Each hash will always contain an error code and message (in english) and may contain a line number and the user uid.

Exceptions:

`Client.AccessDenied`	if the client has no User Administration right.
`Client.WrongParameterException`	if some required parameter is wrongly formed

`Client.IncorrectParameterSyntaxException`	if some required parameter is missing or invalid.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason. In this case, the `errorDetails` hash may contain an `errors` key that holds an array of hashes with the same syntax as the errors contained in a synchronization report result.

2.5.2.13. updateTag(Map<String, Object>, Map<String, Object>)

Updates an existing tag.

Call Example:

```
[
   {
      name => "Bleu",
      domain => "MFT"
   },
   {
      name => "Blue",
      description => "Bleu in english...",
   }
]
```

Expected Argument: Array

id_request (HashTable)

The first argument. Contains the elements that identify the tag in a non ambiguous manner. The HashTable argument must contain the following keys:

`name`, `domain` OR `id` (String, mandatory)	the tag name and domain or internal identifier that identify the tag that should be updated. `name` and `id` are mutually exclusive. When referencing a tag by name, it is mandatory to provide the domain attribute (empty string for a global tag).

update_request (HashTable)

The second argument. Contains a tag definition as defined in Section 2.5.1.1.4, “Tag HashTable”.

Any attribute found in the HashTable and that can be updated will be set to update the tag. Non present attributes will not be modified. To nullify an attribute, it must be present in the HashTable with an empty string.

In the example above, the tag Bleu in domain MFT will have:

- its name changed to "blue"
- its description changed to "Bleu in English..."

Return Value: HashTable

Returns a hash containing the tag information as saved by MFT.

Exceptions:

`Client.AccessDenied`	if the client has no User Administration right.
`Client.WrongParameterException`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

2.5.2.14. updateUser(Map<String, Object>, Map<String, Object>)

Updates an existing user.

Call Example:

```
[
   {
      uid => "jsmith",
      domain => "SmithWorld"
   },
   {
      authentication_policy_alternatemode => "1",
      locale => "es",
      quota_policy => "15MB",
      exchange_policy => ""
   }
]
```

Expected Argument: Array

id_request (HashTable)

The first argument. Contains the elements that identify the user in a non ambiguous manner. The HashTable argument must contain the following keys:

`uid`, `email` OR `id` (String, mandatory)	the user unique identifier, email or internal identifier. The three elements are mutually exclusive.
`domain` (String, optional)	domain of the user (this is needed if the uid or email does not unambiguously identify the user).

update_request (HashTable)

The second argument. Contains a user definition as defined in Section 2.5.1.1.3, “User HashTable”.

Any attribute found in the HashTable and that can be updated will be set to update the user. Non present attributes will not be modified. To nullify an attribute, it must be present in the HashTable with an empty string.

In the example above, the user jsmith in domain SmithWorld will have:

- its locale and quota policy set to "es" and "15MB" respectively
- its alternate authentication mode activated (for the duration configured in the authentication policy)
- its external exchange policy removed (which means he will not be able to exchange with guest users)

Return Value: HashTable

Returns a hash containing the user information as saved by MFT.

Exceptions:

`Client.AccessDenied`	if the client has no User Administration right.
`Client.WrongParameterException`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.

`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

2.5.2.15. updateUsers(Object[], Map<String, Object>)

Updates existing users.

Expected Argument: Array

update (Array)

Contains an array of user update definitions with the same conventions as in `com.opentrust.mft.connector.admin.MFTAdminConnector.updateUser`.

control (HashTable)

Contains some attributes that are used to control how the batch processing must be handled:

`fail_on_error` (String - boolean)	indicates whether creation errors should stop the processing or not. Defaults to `true`.
`return_user_data` (String - boolean)	indicates whether a full user map should be returned for each processed entry, or a simple success/failure information. For performance and if the user data is not used by the caller, this should be set to `false`. Defaults to `true`.

Call Example:

```
First arg:
[
   [
      {
         uid => "jsmith",
         domain => "SmithWorld"
      },
      {
         authentication_policy_alternatemode => "1",
      }
   ],
   [
      {
         id => 5,
      },
      {
         locale => "es",
      }
   ]
]

Second arg:
{
   fail_on_error => 0,
   return_user_data => 1,
}
```

Return Value: HashTable

Returns a hash with the same conventions as `com.opentrust.mft.connector.admin.MFTAdminConnector.createUsers`.

Exceptions:

`Client.AccessDenied`	if the client has no User Administration right.
`Client.WrongParameterException`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

2.5.2.16. version(boolean)

Returns the version information on the connector.

Expected Argument: Array

verbose (String - boolean inputValue)

The only argument. If set to true(1), returns the copyright with version information.

Return Value: HashTable

Returns a hash containing relevant information about the connector and MFT versions.

'api_version' is the Admin Connector API version. 'mft_version' is the version number of the OpenTrust MFT software, and 'mft_revision' is the internal unique version identifier of the software. 'decimal_mft_version' is a decimal number, suitable to make comparisons between versions. Also available are 'jotc_version', 'jotc_revision' and 'decimal_jotc_version', that provide the same information for the Java OTC framework.

if verbose equals 1, an additional key, 'copyright', will be returned, containing a textual copyright information about the product.

"

Chapter 3. Managed File Transfer Rights SOAP/REST Connector Documentation

3.1. Compatibility Note

This documentation is for the version 2.6 of the Managed File Transfer Rights SOAP/REST Connector API. However, the current version of MFT (3.4.0.0) may be backward compatible with previous versions of the API. The following table summarizes the different versions of both the API, and Data marshaling messages supported in this version of MFT:

API Version	MFT Version	API WSDL name space	Message name space	Compatibility
1.1	MFT 3.1.X X >=0 MFT 2.6.X X >=0	http://www.opentrust.com/OpenTrust/JOTC/Connector/Rights/1.1	http://www.opentrust.com/OpenTrust/XML/Message/1.4 (MFT Message)	No
1.0	MFT 1.0.X X >=0 MFT 2.0.X X >=0 MFT 2.1.X X >=0 MFT 2.5.X X >=0	http://www.opentrust.com/OpenTrust/MFT/Connector/Rights/SOAP/1.0	http://www.opentrust.com/OpenTrust/XML/Message/1.3 (MFT Message)	No

3.1.1. API Changes (1.0 to 1.1)

As of MFT 2.6, the Rights Connector is completely handled by the JOTC framework. This has the immediate effect of modifying the WSDL signature and the endpoint of the connector, as described in the table above and in the following sections.

The previous endpoint is still accessible for compatibility reasons, but is deprecated. You are encouraged to migrate your client code to the new WSDL signature as soon as possible. The actual API does not change save for the migration to OTMessage 1.4.

3.2. Network Connection

3.2.1. SOAP Connector

This connector is only accessible via HTTP/S, and the SOAP address location is the following:

`https://mft.example.com/mft/connectors/OTCRightsConnectorSOAPService` to address the current version of the connector

or

`https://mft.example.com/mft/connectors/OTCRightsConnectorSOAPService_X_Y` to address the specific X.Y WSDL namespace

Where *`mft.example.com`* is the hostname of the machine where the MFT Administration Webapp is installed.

The *`mft`* prefix refers to the name of the Administration Webapp.

As an alternative, you can access the connector (API version 1.0) through its previous namespace characteristics at the following location:

`https://mft.example.com/mft/connectors/OTMFTRightsConnectorSOAPService`

3.2.2. REST Connector

This connector is only accessible via HTTP/S, and the base address location is the following:

`https://mft.example.com/mft/connectors/REST/Rights/`

Where *`mft.example.com`* is the hostname of the machine where the MFT Admin Webapp is installed.

The *`mft`* prefix refers to the name of the Administration Webapp.

Considering a function called doStuff, the REST URI for the function will be `https://mft.example.com/mft-user/connectors/REST/Rights/doStuff`.

3.3. List of Connector Operations

This sections lists the methods exposed by the Managed File Transfer Rights SOAP/REST Connector. The methods arguments and returned objects are described in the next chapter.

listGroups	Returns the list of existing groups
getGroups	Returns the list of groups to which the certificate identified by the dn passed as an argument belongs to
getGroupMembers	Returns the list of members and groups that belongs directly to the given groups
addUserToGroup	Adds a certificate to a given group
removeUserFromGroup	Removes a certificate from a given group
addGroupToGroup	Adds a group to another group
removeGroupFromGroup	Removes a group from another group
version	returns version information about the Managed File Transfer Rights SOAP/REST Connector

3.4. Authentication and Access Control

The SOAP client application is authenticated using the processes described in Section 1.4, “Identification”.

Every operation needs the client application to be authenticated.

In order to call any SOAP operation of the Managed File Transfer Rights SOAP/REST Connector, the client application needs to be granted the right Group management on either the Access Control module or on all the groups targeted by the operation (group manager). At least one of the two conditions above must match. See the relevant section of the Managed File Transfer Server Configuration Guide to see how to grant those rights to the user impersonated by the client application.

3.5. Detailed Documentation

3.5.1. API Description

3.5.1.1. API Description

3.5.1.1.1. Name

com.opentrust.otc.connector.rights.RightsConnector - API for interacting with MFT Rights Connector

3.5.1.1.2. Description

com.opentrust.otc.connector.rights.RightsConnector enables execution of programmatically synchronous operations on MFT Rights management system. It exposes methods such as addUserToGroup(), removeUserFromGroup(), etc. It can thus be used to manage rights of individual users and groups (called subjects in this documentation).

The notion of group refers to groups of administrators. Indeed, the rights are those that apply to the MFT Administration operations, and do not concern end-user operations, such as file uploading (those are handled through Exchange Policies applied to a user, with the Admin Connector).

The MFT Rights Connector is a general purpose API that allows an external application to interact with the MFT Rights management.

The main purpose of this class is to be remotely called by a distant client through a dedicated communication layer (for example SOAP). However, the connector does not implement the communication layer.

Note on the internal identifiers

This API uses several internal identifiers for users and groups. Although these identifiers are stored as integers, the caller MUST NOT rely on this internal data type. They must be seen as "opaque" ASCII strings (with NO size limit), that should be used as "handlers" to identify the corresponding objects. Such an identifier is guaranteed to be unique among the type of data (i.e. profile, group...) they identify.

3.5.2. API Methods

3.5.2.1. addGroupToGroup(Map<String, Object>)

Add a group to another group.

Expected Argument: Array

request (HashTable)

The only argument. Contains the elements that identify the groups in a non ambiguous manner. The HashTable argument must contain the following keys:

`group_id` OR `group_name` (String, mandatory)	the identifier or name of the receiver group. The two elements are mutually exclusive.
`included_group_id` OR `included_group_name` (String, mandatory)	the identifier or name of the group to be added to the receiver group. The two elements are mutually exclusive.

Return Value: String

This method returns "1" if the group was added, "0" if the group was already a child. If an error occurred, an exception is thrown.

Exceptions:

`Client.AccessDenied`	if the client has no Group Management right.
`Client.WrongParameterException`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect
`Client.CannotExecuteOperation`	if the operation requested by the client could not be executed for some reason, which has to be specified in the hash contained by `ErrorDetails`. For example, if a group cycle is detected, the reason E_GROUP_CYCLE_DETECTED is specified.

3.5.2.2. addUserToGroup(Map<String, Object>)

Add a user to a given group.

Expected Argument: Array

request (HashTable)

The only argument. Contains the elements that identify the user and group in a non ambiguous manner. The HashTable argument must contain the following keys:

`group_id` OR `group_name` (String, mandatory)	the identifier or name of the group. The two elements are mutually exclusive.
`uid`, `email` OR `id` (String, mandatory)	the user unique identifier, email or internal identifier. The three elements are mutually exclusive.
`domain` (String, optional)	domain of the user (this is needed if the uid or email does not unambiguously identify the user).

Return Value: String

This method returns "1" if the user was added, "0" if the user was already member of the group. If an error occurred, an exception is thrown.

Exceptions:

`Client.AccessDenied`	if the client has no Group Management right.
`Client.WrongParameterException`	if some required method parameter is missing or invalid.

`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

3.5.2.3. getGroupMembers(Map<String, Object>)

Returns the list of members and groups that belong directly to the given group (only members that belong to this group, not members that belong to groups included in this group).

The list only contains the members and groups the principal invoking this method is allowed to see.

Expected Argument: Array

group (HashTable)

The only argument. Contains the elements that identify a group in a non ambiguous manner. The HashTable argument must contain exactly one of the following key:

`group_id` (String)	the group identifier.
`group_name` (String)	the group name.

Return Value: Array of HashTables

This method returns a list of HashTables, which represent the users or groups that directly belong to the group. Each HashTable contains the following keys (and type, between parenthesis):

For users:

`type` (String)	value is always 'USER'.
`uid` (String)	Unique identifier of the user.
`email` (String)	Email of the user.
`id` (String)	Internal identifier of the user.
`domain` (String)	Name of the user's domain.

For groups:

`type` (String)	value is always 'GROUP'.
`group_id` (String)	Internal identifier of the group.
`group_name` (String)	A human readable unique identifier of the group, displayable to a user.
`group_description` (String)	A human readable description of the group, displayable to a user.

If the group does not contain any user or group, the return inputValue will be an empty array.

Exceptions:

`Client.AccessDenied`	if the client has no Group Management right.
`Client.WrongParameterException`	if some required method parameter is missing or invalid.

`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

3.5.2.4. getGroups(Map<String, Object>)

Returns the list of groups to which the given subject belongs.

The list only contains the groups the principal invoking this method is allowed to see. The meaning of "allowed to see" is described in the documentation of `listGroups`.

Expected Argument: Array

subject (HashTable)

The only argument. Contains the elements that identify a subject in a non ambiguous manner. The subject may be a user or group, depending on the content of the argument. The HashTable must contain one of the following keys:

`uid`, `email` OR `id` (String, mandatory)	the user unique identifier, email or internal identifier. The three elements are mutually exclusive.
`domain` (String, optional)	domain of the user (this is needed if the uid or email does not unambiguously identify the user).
`group_id` (String)	the group internal identifier.
`group_name` (String)	the group name.

Return Value: Array of HashTables

This method returns a list of HashTables, which represent the groups the subject belongs to (directly or indirectly). Each HashTable contains the following keys (and type, between parenthesis):

`type` (String)	value is always 'GROUP'.
`group_id` (String)	Internal identifier of the group.
`group_name` (String)	A human readable unique identifier of the group, displayable to a user.
`group_description` (String)	A human readable description of the group, displayable to a user.

If the subject does not belong to any group, the return inputValue will be an empty array.

Exceptions:

`Client.AccessDenied`	if the client has no Group Management right.
`Client.WrongParameterException`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

3.5.2.5. listGroups()

Returns the list of existing groups the principal is allowed to see.

The principal invoking this method is allowed to see a group if at least one of the conditions is verified:

- he possesses the global right "Group Management"
- he is a manager of the group
- he is a member of the group, either directly or indirectly via other nested groups

Expected Argument: none

Return Value: Array of HashTables

This method returns a list of HashTables, which represent the available groups. Each HashTable contains the following keys (and type, between parenthesis):

`type` (String)	value is always 'GROUP'.
`group_id` (Internal idenfier, String)	Internal identifier of the group.
`group_name` (String)	A human readable unique identifier of the group, displayable to a user.
`group_description` (String)	A human readable description of the group, displayable to a user.
`group_management`(String)	Value is '1' if the administrator that requested this list has rights to manage users from the group, '0' either.

If there is no group, the return inputValue will be an empty array.

3.5.2.6. removeGroupFromGroup(Map<String, Object>)

Removes a group from another group.

Expected Argument: Array

request (HashTable)

The only argument. Contains the elements that identify the groups in a non ambiguous manner. The HashTable argument must the following keys:

`group_id` OR `group_name` (String, mandatory)	the identifier or name of the receiver group. The two elements are mutually exclusive.
`included_group_id` OR `included_group_name` (String, mandatory)	the identifier or name of the group to be removed from the receiver group. The two elements are mutually exclusive.

Return Value: String

This method returns "1" if the group was removed, "0" if the group was not a child. If an error occurred, an exception is thrown.

Exceptions:

`Client.AccessDenied`	if the client has no Group Management right.
`Client.WrongParameterException`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	If the syntax of one of the arguments is incorrect

`Client.CannotExecuteOperation`	if the operation requested by the client could not be executed for some reason.

3.5.2.7. removeUserFromGroup(Map<String, Object>)

Removes a user from a given group.

Expected Argument: Array

request (HashTable)

The only argument. Contains the elements that identify the group and user in a non ambiguous manner. The HashTable argument must the following keys:

`group_id` OR `group_name` (String, mandatory)	the identifier or name of the group. The two elements are mutually exclusive.
`uid` OR `email` (String, mandatory)	the user unique identifier or email. The two elements are mutually exclusive.
`domain` (String, optional)	domain of the user (this is needed if the uid or email does not unambiguously identify the user).

Return Value: String

This method returns "1" if the user was removed, "0" if the user was not a member of the group. If an error occurred, an exception is thrown.

Exceptions:

`Client.AccessDenied`	if the client has no Group Management right.
`Client.WrongParameterException`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect
`Client.CannotExecuteOperation`	if the operation requested by the client could not be executed for some reason.

3.5.2.8. version(boolean)

Returns the version information on the connector.

Expected Argument: Array

verbose (String - boolean inputValue)

The only argument. If set to true(1), returns the copyright with version information.

Return Value: HashTable

Returns a hash containing relevant information about the connector and MFT versions.

'api_version' is the Rights Connector API version. 'mft_version' is the version number of the MFT software, and 'app_revision' is the internal unique version identifier of the software. 'decimal_mft_version' is a decimal number, suitable to make comparisons between versions. Also available are 'jotc_version', 'jotc_revision' and 'decimal_jotc_version', that provide the same information for the Java OTC framework.

if verbose equals 1, an additional key, 'copyright', will be returned, containing a textual copyright information about the product.

Chapter 4. Managed File Transfer File SOAP/REST Connector Documentation

4.1. Compatibility Note

This documentation is for the version 2.6 of the Managed File Transfer File SOAP/REST Connector API. However, the current version of MFT (3.4.0.0) may be backward compatible with previous versions of the API. The following table summarizes the different versions of both the API, and Data marshaling messages supported in this version of MFT:

API Version	MFT Version	API WSDL name space	Message name space	Compatibility
2.6	MFT 3.1.X X >=0 MFT 2.6.X X >=0	http://www.opentrust.com/OpenTrust/MFT/Connector/File/SOAP/1.1	http://www.opentrust.com/OpenTrust/XML/Message/1.4 (MFT Message)	Yes
2.0	MFT 2.5.X X >=0 MFT 2.1.X X >=0 MFT 2.0.X X >=0	http://www.opentrust.com/OpenTrust/MFT/Connector/File/SOAP/1.0	http://www.opentrust.com/OpenTrust/XML/Message/1.3 (MFT Message)	Yes

4.1.1. API Changes (2.0 to 2.6)

This section lists the changes between the versions 2.0 and 2.6 of the Managed File Transfer File SOAP/REST Connector that may have an impact on the use of the connector. Changes that do not have a compatibility impact are not listed. The changes are listed in the following table:

Operations	Changes
all	The migration to OTMessage 1.4 triggered a change in the WSDL namespace of the Managed File Transfer File SOAP/REST Connector and an API version bump.
listMessages	The recipients attribute has changed: The returned value is now an array of strings representing the recipients email. In previous versions, the returned value was a string representing the comma separated list of the recipients emails.

4.2. Network Connection

4.2.1. SOAP Connector

This connector is only accessible via HTTP/S, and the SOAP address location is the following:

`https://mft.example.com/zephyr/connectors/OTMFTFileConnectorSOAPService` to address the current version of the connector

or

`https://mft.example.com/zephyr/connectors/OTMFTFileConnectorSOAPService_X_Y` to address the specific X.Y WSDL namespace

Where *`mft.example.com`* is the hostname of the machine where the MFT User Webapp is installed.

The *`zephyr`* prefix refers to the name of the User Webapp. It may be different depending on your installation setup.

4.2.2. REST Connector

This connector is only accessible via HTTP/S, and the base address location is the following:

`https://mft.example.com/zephyr/connectors/REST/`

Where *`mft.example.com`* is the hostname of the machine where the MFT User Webapp is installed.

The *`zephyr`* prefix refers to the name of the User Webapp. It may be different depending on your installation setup.

Considering a function called doStuff, the REST URI for the function will be `https://mft.example.com/zephyr/connectors/REST/doStuff`.

4.3. List of Connector Operations

This sections lists the methods exposed by the Managed File Transfer File SOAP/REST Connector. The methods arguments and returned objects are described in the next chapter.

createUploadToken	Create a new upload token.
deleteMessage	Delete a message sent by the authenticated user.
deleteUploadToken	Delete an existing upload token.
getContext	Get information on the authenticated user.
getMessage	Get information on a message.
getMessageUrls	Get access and download URLs for a message.
getRecipientInfo	Get information about a recipient.

getUploadToken	Get information about an existing upload token.
listMessages	List all available messages for the authenticated user.
listProjects	List all available projects for to the authenticated user.
listUploadTokens	List all upload tokens created by the authenticated user.
sendMessage	Send a new message (using files already uploaded on the MFT platform).
updateMessage	Update a message (add recipients or extend the lifetime).
updateUploadToken	Update an existing upload token.
version	return version information about the Managed File Transfer File SOAP/REST Connector

4.4. Authentication and Access Control

The client application is authenticated using the processes described in Section 1.4, “Identification”.

Every operation needs the client application to be authenticated.

To access the File SOAP/REST Connector the client application must be granted the explicit privilege to do so using an Exchange Policy (this is set in the MFT user information panel).

4.5. Detailed Documentation

4.5.1. API Description

4.5.1.1. API Description

4.5.1.1.1. Name

com.opentrust.mft.connector.file.FileConnector - API for interacting with OpenTrust MFT File Connector

4.5.1.1.2. Description

com.opentrust.mft.connector.file.FileConnector enables execution of synchronous operations programmatically on the OpenTrust MFT User servers. It exposes methods such as sendMessage(), getMessage(), etc, that make it possible to send or download messages through programmatic interfaces and standard (FTP, for example) file transfer methods instead of the OpenTrust MFT Web UI.

The OpenTrust MFT File Connector is a general purpose API that allows an external application to interact with the MFT User servers.

The main purpose of this class is to be remotely called by a distant client through a dedicated communication layer (for example SOAP). However, it does not implement the communication layer.

Note on the internal identifiers

This API uses several internal identifiers for messages, projects or profiles. Although these identifiers are stored as integers, the caller MUST NOT rely on this internal data type. They must be seen as "opaque" ASCII strings (with NO size limit), that should be used as "handlers" to identify the corresponding objects. Such an identifier is guaranteed to be unique among the type of data (i.e. profile, group..) they identify.

4.5.1.1.3. Message HashTable

In method calls, a message is described using a `otmsg:HashTable` with the following structure:

```
 {
  id => "4",
  type => "simple",
  subject => "Very Important Document",
  sender => {
                email  => "john.smith@acme.ltd",
                uid    => "jsmith",
                domain => "MFT"
            },
  recipients => [
                    {
                       index  => "0",
                       email  => "john.doe@acme.ltd",
                       uid    => "jdoe",
                       domain => "MFT"
                    },
                    {
                       index  => "1",
                       email  => "jane.smith@not-acme.ltd",
                    }
                ],
  date => "20100325122535Z",
  expiration_date => "20101525122535Z",
  comment => "This document must be read by all recipients before today at no\
on. It is mandatory that every participant to this afternoon\'s meeting be \
up to date.",
  encrypted => 1,
  signed => 0,
  pre_archiving_status => "NO_PREARCHIVING",
  active => 1,
  files => [
               {
                 index         => "0",
                 name          => "Meeting Preview.doc",
                 size          => "123340",
                 download_url  => "https://myserver/zephyr/connectors/REST/d\
ownloadFile?file=30&os=linux",
                 digest        => "267bb122e262f44cd15de498be839b7c0a880afd7\
f6187ea6df1e3940b56511e"
               },
               {
                 index         => "1",
                 name          => "Meeting Presence Sheet.xls",
                 size          => "1354",
                 download_url  => "https://myserver/zephyr/connectors/REST/d\
ownloadFile?file=31&os=linux",
                 digest        => "267bb122e262f44cd15de498be839b7c0a880afd7\
f6187ea6df1e3940b56511e"
               },
               {
                 index         => "2",
                 name          => "Vacation Pictures.zip",
                 size          => "11453145",
                 download_url  => "https://myserver/zephyr/connectors/REST/d\
ownloadFile?file=33&os=linux",
                 digest        => "267bb122e262f44cd15de498be839b7c0a880afd7\
f6187ea6df1e3940b56511e"
               }
            ],
   download_url => "https://myserver/zephyr/connectors/REST/downloadFile?mess\
age=34&os=linux",
   access_url => "https://myserver/zephyr/WebApp.jsp?ht=tab-MESSAGES-DETAILS-\
4",
   size => "11577839",
   audit => {
              viewed => [
                           {
                              user_index => "0",
                              date => "20100325122935Z"
                           },
                           {
```

```
                              user_index => "1",
                              date => "20100325132535Z"
                           },
                        ],
               downloaded => [
                              {
                                 user_index => "1",
                                 file_index => "0",
                                 date => "20100325132554Z"
                              },
                              {
                                 user_index => "1"
                                 file_index => "2",
                                 date => "20100325132554Z"
                              },
                              {
                                 user_index => "1",
                                 file_index => "3",
                                 date => "20100325132554Z"
                              },
                           ]
            }
}
```

The attributes are detailed below:

`id` (String - long value)	internal message identifier. This id is managed internally by MFT.
`type` (String)	type of message, one of simple or project.
`subject` (String, max 64 characters)	the message's subject.
`sender` (HashTable)	the message's sender. This is an HashTable as defined in Actor HashTable.
`recipients` (Array)	the message's recipients. This is an Array of HashTable as defined in Actor HashTable. This information is present even when the message is sent to a Project, and provides the actual list of recipients to this message.
`project` (HashTable)	the message's project, when applicable. This is an HashTable with the following keys:
	`project_id` (String - long value) — the project's internal identifier, as returned by the `listProjects` method.
	`project_name` (String) — the project's name.
`date` (String - date value)	the message's creation date.
`expiration_date` (String - date value)	the message's expiration date (computed as the message's creation date, plus its lifetime in days).
`active` (String - boolean value)	whether the message files are still available for download or not.
`comment` (String, maximum 2048 characters)	the message's comment
`encrypted` (String - boolean value)	whether the message files are encrypted or not. When this is the case, they are consolidated in a ZIP Archive, and there is only one file in the message.
`signed` (String - boolean value)	whether the message files are signed or not.

`pre_archiving_status` (String)	the pre-archiving status. Can be one of the following : NO_PREARCHIVING, TO_BE_PREARCHIVED, PREARCHIVED, ERROR
`files` (Array)	array of HashTables that describe files attached to this message. These HashTables contain the following attributes:
	`index` (String - long value) — index of the file in the list of files. This index is local to the current message, and will be used to reference this file in other parts of the message.
	`name` (String) — the file name (without a path), including the extension.
	`size` (String) — the file size, in octets.
	`download_url` (String) — the file direct download URL.
	`digest` (String) — the SHA256 digest value of the file.
`download_url` (String)	the direct download URL of all the message files. This will download the message file if there is only one, or a ZIP archive containing all of the message files if there are several.
`size` (String)	the whole message size (the sum of the sizes of all attached files)
`audit` (HashTable)	audit information about the message, contains two attributes, viewed and downloaded.
	`viewed` (HashTable) — list of recipients that have viewed the message. The recipient is referenced using his index, and the viewing date is also given (date of first view).
	`downloaded` (HashTable) — list of recipients that have downloaded file, for each file. The recipient and file are referenced using their index, and the download date is also given (date of first download).

Notes:

- When the message was sent to a project, the recipients attribute is replaced by a project attribute.

4.5.1.1.4. Upload Token HashTable

In method calls, an Upload Token is described using a `otmsg:HashTable` with the following structures:

```
{
   token_value  => "14ueuvop5ojk3bgblun7upt06k1m9io12dfh4m6sin1j9jeaii76",
   creator  => {
                email  => "john.smith@acme.ltd",
                uid    => "jsmith",
                domain => "MFT",
```

```
                },
    email => "jane.doe@another-acme.ltd",
    creation_date    => "20100726081111Z",
    comment => "This is an upload token for you personal usage.",
    expiration_date  => "20100729081111Z",
    lifetime => 3,
    max_messages => 5,
    quota => 10,
    message_count => 2,
    access_url => "http://mft.opentrust.com/zephyr/UploadToken.jsp?token=14ue\
uvop5ojk3bgblun7upt06k1m9io12dfh4m6sin1j9jeaii76"
 }
```

The attributes are detailed below:

`token_value` (String)	the value of the token.
`creator` (String, readonly)	information about the token's creator, organized as an Section 4.5.1.1.5, “Actor HashTable” HashTable.
`email` (String)	the token's recipient email.
`creation_date` (String, readonly)	the token's creation date.
`comment` (String)	The token's comment text.
`expiration_date` (String, readonly)	The token's expiration date (i.e. the creation date plus the token's lifetime in days).
`lifetime` (String, integer value)	The token's lifetime, in days.
`max_messages` (String - integer value)	The maximum number of messages that can be sent using this token. If set to 0, then there are no limitations.
`quota` (String - integer value)	The token's quota, in megabytes. If set to 0 then there is no quota (in the limits of the creator's own quota).
`message_count` (String, readonly)	The number of messages that have been sent using this token.
`access_url` (String, readonly)	The token's access URL, used to access the web interface with this token.

4.5.1.1.5. Actor HashTable

In method calls, a message actor (sender or recipient) is described using a `otmsg:HashTable` with the following structures:

```
Registered User:
{
   index  => "0",
   email  => "john.smith@acme.ltd",
   uid    => "jsmith",
   domain => "MFT",
   type   => "registered"
}

Guest User:
{
   index  => "1",
   email  => "jane.do@not-acme.ltd",
   type   => "guest"
}
```

The attributes are detailed below:

`index` (String - long value)	index of the actor in a list of actor. This is optional and may not appear depending on context. When it appears, it must be considered local to the current message, and will be used to reference this actor in other parts of the message.
`email` (String)	the actor's email.
`uid` (String)	the user's unique identifier (registered users only), as defined in his MFT account.
`domain` (String)	the user's MFT domain name (registered users only).
`type` (String)	type of Actor, one of 'registered' or 'guest'. Depending on context, this attribute may not be present.

4.5.2. API Methods

4.5.2.1. createUploadToken(Map<String, Object>, boolean)

Creates a new Upload Token.

REST support: JSON and simple query

Expected Argument: Array

request (HashTable)

The first argument. Contains an Upload Token definition as defined in Section 4.5.1.1.4, "Upload Token HashTable". Only the following keys can be present:

`email` (mandatory)

`lifetime` (mandatory)

`max_messages` (mandatory)

`quota` (optional)

`comment` (optional)

returnTokenValueOnly (String, optional - boolean value)

The second argument. This is optional and contains a boolean value that indicates whether the method should return a full Upload Token detail HashTable or simply the token value (as a String). By default, the method returns a full UploadToken detail HashTable.

Tip: when using the REST interface, this second parameter can be passed using the `return_token_value` attribute in the first argument's HashTable. This allows requesting a single token value while still making a simple query REST call.

Return Value: HashTable

Returns a hash containing the token information as saved by MFT.

Exceptions:

`Client.AccessDenied`	if the client has no token creation privilege.
`Client.WrongParameterException`	if some parameter is missing.

`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

4.5.2.2. deleteMessage(Map<String, Object>)

Delete a message.

REST support: JSON and simple query

Expected Argument: Array

request (HashTable)

The only argument. Identify the message that should be deleted. This contains one of the following attributes:

`id` (String - long value)	message internal identifier as returned by the `listMessages` operation.
`ids` (Array)	array of Strings (as long values) that provides the internal identifiers of the message that are to be deleted.

The `id` and `ids` attributes are mutually exclusive.

Return Value: String

This method returns "1" on success or throws an error (see below).

Exceptions:

`Client.AccessDenied`	if the client cannot delete the defined message.
`Client.WrongParameter`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason (namely message not found or already deleted)

4.5.2.3. deleteUploadToken(Map<String, Object>)

Deletes an existing Upload Token, removing the usage privileges from the recipient. Only the token creator can delete it.

REST support: JSON and simple query

Expected Argument: Array

request (HashTable)

The only argument. Contains a HashTable with the `token_value` attribute that identifies the token that must be deleted.

Return Value: String

This method returns "1" on success or throws an error (see below).

Exceptions:

`Client.AccessDenied`	if the client is not the token's creator.

`Client.WrongParameter`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason (namely upload token not found or already deleted).

4.5.2.4. getContext()

Returns context information about the authenticated user, providing information on its attached policies.

REST support: JSON and simple query

Expected Argument: none

Return Value: HashTable

The returned HashTable contains information about the user, with the following structure:

```
{
   "uid" => "jsmith",
   "first_name" => "John",
   "last_name" => "smith",
   "email" => "jsmith@acme.ltd",
   "domain" => "MFT",
   "quota" => {
         "total" => 1048576,
         "remaining" => 245024,
         "sending" => 34556,
   },
   "sending_policy" => {
         "id" => 1423,
         "name" => "My user sending policy",
         "message_maxsize" => 2048576,
         "antivirus" => 1,
         "antivirus_maxsize" => 1048576,
         "lifetime_fixed" => 0,
         "lifetime_default" => 10,
         "lifetime_max" => 50,
         "encrypt_mode" => "OPTIONAL_NOT_SUGGESTED",
         "encrypt_maxsize" => 1048576,
         "pdf_signature_mode" => "DISABLED",
         "pre_archiving_mode"  => "DISABLED"
   }
   "exchange_policy" => {
         "can_send_messages" => 1,
         "max_guest_recipients" => 5,
         "can_create_upload_tokens" => 1,
         "max_upload_token_lifetime" => 90,
         "registered_users_rule" => {
             "name" => "Own Domain and all emails",
             "domains_rule" => "OWN_USER_DOMAIN",
             "email_domains_rule" => "ALL_DOMAINS",
         },
         "guest_users_rule" => {
             "name" => "Custom emails",
             "email_domains_rule" => "CUSTOM",
             "email_domains" => [ "@acme.ltd", "@*.acme.com" ]
         }
   }
}
```

The encrypt_mode property can have the following values: DISABLED, OPTIONAL_NOT_SUGGESTED, OPTIONAL_SUGGESTED, MANDATORY.

The pdf_signature_mode property can have the following values: DISABLED, OPTIONAL, MANDATORY.

The pre_archiving_mode property can have the following values: DISABLED, OPTIONAL_NOT_SUGGESTED, OPTIONAL_SUGGESTED, MANDATORY.

The domains_rule and email_domains_rule property can have the following values: ALL_DOMAINS, OWN_USER_DOMAIN, CUSTOM, NO_ACCESS.

Exceptions:

`Client.AccessDenied`	if the client cannot open the session.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

4.5.2.5. getMessage(Map<String, Object>)

Returns information about a message.

REST support: JSON and simple query

Expected Argument: Array

request (HashTable)

The only argument. Must contain the following keys:

`id` (String - long value)	the message's internal identifier, as returned by the `listMessages` operation.

Also, the request may contain the following keys:

`with_audit` (String - boolean value) defaults to **`false`**	whether to include audit information in the message details or not.
`operating_system` (String) defaults to **`windows`**	what operating system to target for encrypted files downloads. Either 'windows', 'linux' or 'mac'.

Return Value: HashTable

Returns a message definition as defined in Section 4.5.1.1.3, “Message HashTable”.

Exceptions:

`Client.AccessDenied`	if the client cannot access the message's details.
`Client.WrongParameter`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

4.5.2.6. getMessageUrls(Map<String, Object>)

Returns the URLs that may be used to access or download the message, depending on the recipients. This method can only be called for a message whose creator is the connecting user.

REST support: JSON and simple query

Expected Argument: Array

request (HashTable)

The only argument. Must contain the following keys:

`id` (String - long value)	the message's internal identifier, as returned by the `listMessages` operation.

`operating_system` (String) defaults to **`windows`**	what operating system to target for encrypted files downloads. Either 'windows', 'linux' or 'mac'.

Return Value: HashTable

The returned HashTable contains all of the access and download URL to the message, for each recipient. The HashTable has the following structure:

```
 {
    "jsmith@acme.ltd"   => {
             type => "registered",
             uid => "jsmith",
             domain => "ACME",
             access_url => "https://myserver/zephyr/WebApp.jsp#tab-MESSAGES-D\
ETAILS-34",
             download_url => "https://myserver/zephyr/connectors/REST/downloa\
dFile?message=34&os=linux"
             },
    "jdoe@not-acme.ltd" => {
             type => "guest",
             access_url => "https://myserver/zephyr/DownloadToken.jsp?token=7\
4cflajao07hd9hctqqe2alt1j5nitl3maohea27smdsm7vrmf3",
             download_url => "https://myserver/zephyr/connectors/REST/downloa\
dFile?token=74cflajao07hd9hctqqe2alt1j5nitl3maohea27smdsm7vrmf3&message=34&os\
=linux"
             }
 }
```

The HashTable keys are the email of each of the message's recipients. Each key has a HashTable value that gives the access and download (all files as a ZIP archive) URLs for the corresponding recipient. It contains the following keys:

`type` (String)	type of recipient, either `registered` or `guest`.
`uid` (String)	the uid of the recipient, set only when type is `registered`.
`domain` (String)	the domain of the recipient, set only when type is `registered`.
`access_url` (String)	the recipient's access URL to the message.
`download_url` (String)	the recipient's download URL to the message's files (in a ZIP archive if there are several files).

Exceptions:

`Client.AccessDenied`	if the client cannot access the message's urls.
`Client.WrongParameter`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

4.5.2.7. getRecipientInfo(Map<String, Object>)

Resolve an email address into a recipient in the context of a message creation.

REST support: JSON and simple query

Expected Argument: Array

request (HashTable)

The only argument. Contains a HashTable with the `email` attribute that identifies a recipient.

Return Value: HashTable

Returns a hash containing the recipient information as defined in Section 4.5.1.1.5, “Actor HashTable” or an empty hash if the email address cannot be used to create a message (exchange policy restriction).

Exceptions:

`Client.AccessDenied`	if the client is not the token's creator.
`Client.WrongParameter`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

4.5.2.8. getUploadToken(Map<String, Object>)

Returns information about an Upload Token.

REST support: JSON and simple query

Expected Argument: Array

request (HashTable)

The only argument. Contains a HashTable with the `token_value` attribute that identifies the token.

Return Value: HashTable

Returns a hash containing the token information as defined in Section 4.5.1.1.4, “Upload Token HashTable”.

Exceptions:

`Client.AccessDenied`	if the client is not the token's creator.
`Client.WrongParameter`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

4.5.2.9. listMessages()

List the active messages of the connecting user. Only those messages whose files are available for download are listed.

Expected Argument: none

Return Value: Array

Returns the list of available messages for the connecting user as an array of HashTable objects with the following structure:

```
[
   {
      message_id => "26",
      subject => "Very important files",
      creation_date => "20101006164010Z",
      expiration_date => "20101016164010Z",
      sender => "john.doe@acme.ltd",
      recipients => [ "jane.doe@acme.ltd", "paul.smith@another-acme.ltd" ],
```

```
      nb_files => "2",
      viewed => "1",
      sent => "1"
   },
   {
      message_id => "15",
      subject => "Relatively important files",
      creation_date => "20101002164010Z",
      expiration_date => "20101012164010Z",
      sender => "john.doe@acme.ltd",
      project => "VIP Workgroup",
      project_id => "4",
      nb_files => "2",
      viewed => "1",
      sent => "1"
   },
   {
      message_id => "10",
      subject => "Not important at all",
      creation_date => "20101006164010Z",
      expiration_date => "20101016164010Z",
      sender => "jane.smith@acme.ltd",
      viewed => "0",
      sent => "0"
   },
]
```

Each message entry contains the following attributes:

`message_id` (String - long value)	message internal identifier, may be used in methods such as `getMessage`.
`subject` (String)	message subject
`creation_date` (String - date value)	message creation date
`expiration_date` (String - date value)	message expiration date
`sender` (String)	email of the message's sender
`recipients` (Array)	array of strings representing the recipients email (this attribute is mutually exclusive with project/project_id, see below)
`project` (String)	when the message is in a project, the project name
`project_id` (String - long value)	when the message is in a project, the project internal identifier
`nb_files` (String - long value)	the number of files contained in the message
`viewed` (String - boolean value)	whether the message has been flagged as 'viewed' for the caller (this is always true for the sender)
`sent` (String - boolean value)	whether the message has been sent by the caller

Note: the list of recipients attribute is set only when the connecting user is the sender of the message.

Exceptions:

`Client.AccessDenied`	if the client cannot access the operation.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

4.5.2.10. listProjects(boolean)

List the configured projects that are accessible to the connecting user.

Expected Argument: Array

verbose (optional) (String - boolean inputValue)

* EXPERIMENTAL * Tells whether or not to return information about the sending policy of the projects, using the `sending_policy` key. The structure is the same as the one used in `getContext` .

Return Value: Array

Returns the list of available projects for the connecting user as an array of HashTable objects with the following structure:

```
[
   {
      project_id => "2415",
      project_name => "My Super Project",
   },
   {
      project_id => "1245",
      project_name => "My Other Project",
   },
]
```

Exceptions:

`Client.AccessDenied`	if the client cannot access the operation.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

4.5.2.11. listUploadTokens()

List the active Upload Tokens that belong to the connecting user.

Expected Argument: none

Return Value: Array

Returns the list of Upload Tokens created by the connecting user as an array of HashTable objects with the following structure:

```
[
   {
      token_value => "1oeuo59tlrikb9rsva669gep8ks1bm1j0oa4mqaj3n1ug3horcp2",
      email => "john.doe@acme.ltd",
      expiration_date => "20101016141448Z"
   },
   {
      token_value => "1st2l0pl5o4m5cecovrl835hu3eto5r9u4okokvholsioniuruoj",
      email => "jane.doe@acme.ltd",
      expiration_date => "20101016141448Z"
   },
]
```

4.5.2.12. sendMessage(Map<String, Object>)

Sends a message with files either already deposited on the MFT platform's file system ("offline" sending mode) or directly sent through the request ("online" sending mode).

REST support: JSON and simple query. Simple query is mandatory when sending a message with online files.

Expected Argument: Array

request (HashTable)

The only argument. Must contain the following keys:

`recipients` (Array) or `project_id` (String)	Depending on whether this is a simple message or a message sent to a project: the list of recipients, as email addresses, or the project internal identifier (as returned by `listProjects`).
`files` (Array of HashTables - see structure below)	"offline" mode only, a description of the files to send.

It may contain the following keys:

`subject` (String)	the message subject. If missing or empty, the name of the first file is used.
`comment` (String)	the message comment.
`lifetime` (String - int value) defaults to sending policy's default lifetime value	the message's lifetime, in days.
`encrypted` (String - boolean value) defaults to **`false`**	whether to encrypt the file(s) or not.
`password` (String)	encryption password, when encrypted is true.
`file_number` (String - int value) Online mode only	the number of uploaded files. This parameter is only required in online mode when encrypted is true and more than one file is uploaded. It gives the server a hint about the necessity to generate on the fly a ZIP archive containing the files (the ZIP archive is eventually encrypted). If this parameter is missing in online mode when multiple files to encrypt are uploaded, the operation will fail.
`signed` (String - boolean value) defaults to **`false`**	whether to sign the PDF file(s) or not.
`pre_archiving_requested` (String - boolean value) defaults to **`false`**	whether to pre-archive the file(s) or not.
`testrun` (String - boolean value) defaults to **`false`**	make a "testrun" call where the parameters are only tested for validity, but no actual message is sent. The `files` parameter must still be present but will be ignored in this mode.

The HashTable has the following structure:

```
 {
  subject => "Very Important Document",
  recipients => [ "jane.smith@not-acme.ltd", "john.doe@acme.ltd" ],
  comment    => "This document must be read by all recipients before today at\
 noon. It is mandatory that every participant to this afternoon\'s meeting \
be up to date.",
  lifetime   => "10",
  encrypted  => 1,
  signed  => 0,
  password   => "somesecret",
  files => [
              {
                 name => "Meeting Preview.doc",
                 digest => "a3ab0473b1681c8a20d6c518b97ad044eae67309dc77e1b48\
b7398b9b5d937da"
              },
              {
                 name => "Meeting Presence Sheet.xls",
                 digest => "2cecc8c283997ec2ec5b18d2ab34ab498ff92de2690bd7885\
a7cf3b92878287e"
              }
```

```
        ],
}
```

Important, this method can be called using two different modes:

Online mode (REST only)	In this mode, the files are uploaded directly as part of the HTTP request (multipart POST request). The `files` attribute is ignored if present. See Section 4.5.4.3, "Online Mode".
Offline mode (SOAP and REST)	In this mode, the files must already be available to the MFT platform on the server's file system. See Section 4.5.4.2, "Offline Mode".

For more information on how files must be sent to the server, please refer to Section 4.5.4, "File Uploads".

Notes:

- The recipients attribute contains the list of the recipient's email addresses. These are automatically extrapolated to registered users or guest users depending on the sender's authorizations.
- When encrypted is `false`, the password attribute should not be set (it will be ignored).
- the digest attribute is optional (remember the `files` attribute only makes sense in Offline Mode). If provided, it must contain the hex representation of the SHA-256 digest of the file already deposited on the server file system; it thus ensures that the given file reference really matches the file deposited on disk that will be processed by the MFT server. If the digest is not provided, no digest check is done, and only the provided file `name` will be used to match the file deposited on disk.
- If all of the recipients email addresses are invalid or non authorized, then the whole send operation fails. The send operation succeeds if there is at least one valid recipient.
- If one of the request parameters cannot be respected because the server configuration forbids it, then the operation fails (for example if the PDF signature is requested but not configured on the server).

Return Value: HashTable

When a nominal call is made, returns a message definition as defined in Section 4.5.1.1.3, "Message HashTable". This is the definition of the message that was just sent, after it has been processed by the MFT platform. The following additional parameters may be present:

`forbidden_emails` (Array of Strings)	the list of emails that were forbidden amongst the recipients. The message was sent, but not to these recipients.
`wrong_syntax_emails` (Array of Strings)	the list of emails whose syntax was invalid, amongst the recipients. The message was sent, but not to these recipients.

When a "test run" call is made, returns the following structure (unless an error occurs):

```
{
  'testrun' => '1'
}
```

Exceptions:

`Client.AccessDenied`	if the client cannot send the defined message.
`Client.WrongParameter`	if some required method parameter is missing or invalid.

`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

4.5.2.13. updateMessage(Map<String, Object>)

Update a message: add recipients, extend the lifetime.

REST support: JSON only

Expected Argument: Array

request (HashTable)

The only argument. Describe the update operations with the following structure:

```
{
   id => "144566",
   recipients_add => [ "jane.doe@not-acme.ltd", "john.doe@acme.ltd" ]
   lifetime_extend => "3",
}
```

The `id` attribute is mandatory: it references a message internal identifier as returned by the `listMessages` operation.

Also, the argument must contain at least one of the attributes detailed below:

`recipients_add` (Array)	list of recipients to add. The recipients are defined as an Array using the same conventions as `sendMessage`.
`lifetime_extend` (String - long value)	number of days the message lifetime will be extended with.

Return Value: HashTable

Returns a message definition as defined in Section 4.5.1.1.3, "Message HashTable". This is the definition of the message that was just updated, after it has been processed by the MFT platform.

Exceptions:

`Client.AccessDenied`	if the client cannot update the defined message.
`Client.WrongParameter`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	if the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

4.5.2.14. updateUploadToken(Map<String, Object>)

Updates an existing Upload Token.

REST support: JSON and simple query

Call Example:

```
{
   token_value => "1rlvdo7kktful0tlepgjpms9h8seont3mupbdgop9l6rsmfed6hc",
```

```
    quota => "15",
    max_messages => "3",
    lifetime => "99",
    comment => "Restrained token usage to 15MB, 3 messages max and 99 days li\
fetime"
 },
```

Expected Argument: Array

request (HashTable)

The only argument. Contains a HashTable with the `token_value` attribute that identifies the token that must be updated, and any of the following attributes:

quota

max_messages

lifetime

comment

The token will be updated to match the new attributes values. Non present attributes will not be touched. To set an unlimited quota or to remove any limitation to the maximum number of messages, set these values to 0.

In the example above, the token 1rlvdo7kktful0tlepgjpms9h8seont3mupbdgop9l6rsmfed6hc will have:

- its quota set to 15MB
- its maximum number of messages set to 3
- its lifetime set to 99 days
- its comment is updated

Return Value: HashTable

Returns a hash containing the Upload Token information as saved by MFT.

Exceptions:

`Client.AccessDenied`	if the client is not the token's creator.
`Client.WrongParameterException`	if some required method parameter is missing or invalid.
`Client.IncorrectParameterSyntax`	If the syntax of one of the arguments is incorrect or if an argument is missing.
`Client.CannotExecuteOperation`	if the operation could not be completed for some reason.

4.5.2.15. version(boolean)

Returns the version information on the connector.

Expected Argument: Array

verbose (String - boolean inputValue)

The only argument. If set to true(1), returns the copyright with version information.

Return Value: HashTable

Returns a hash containing relevant information about the connector and MFT versions.

'api_version' is the File Connector API version. 'mft_version' is the version number of the OpenTrust MFT software, and 'mft_revision' is the internal unique version identifier of the software. 'decimal_mft_version' is a decimal number, suitable to make comparisons between versions. Also available are 'jotc_version', 'jotc_revision' and 'decimal_jotc_version', that provide the same information for the Java OTC framework.

if verbose equals 1, an additional key, 'copyright', will be returned, containing a textual copyright information about the product.

"

4.5.3. File Downloads

Some File SOAP/REST Connector operations return download URLs that can be used by registered users of MFT. However, to access the files, authentication credentials must be provided in addition to the URL's HTTP parameters.

These credentials are HTTP parameters or headers that are named according to the authentication configuration of the application and the user. Typically, the rules defined in Client Authentication should be followed.

However, an URL with a Download Token can be used "as is".

Example:

The following download URL

```
https://myserver/zephyr/connectors/REST/downloadFile?file=30&os=linux
```

should be used in an HTTP request with the `OTC-Auth-Uid` and `OTC-Auth-Password` HTTP headers set.

> **Important**
>
> The given URLs can be used with GET or POST HTTP commands.

4.5.4. File Uploads

4.5.4.1. Introduction

Files sent using the Managed File Transfer File SOAP/REST Connector sendMessage method can be uploaded in two ways:

- Files are sent directly within the connector's HTTP Request (Online Mode)
- Files are first uploaded on the MFT platform, then a call to the connector is made that references these files (Offline Mode)

> **Important**
>
> The Online Mode is only available while using the REST connector interface.

Important

Online Mode is assumed whenever the connector receives a multipart form POST request. If a REST client application uses the Managed File Transfer File SOAP/REST Connector to send messages in both Online and Offline Mode, it must make sure that requests to send messages in Offline Mode use a standard form POST or GET (not a multipart).

Although the semantics of each call are mostly the same, there are some limitations and/or specific behaviours depending on the file upload mode that is chosen. More specifically, using the Offline Mode is pretty straightforward and similar to using any of the others Managed File Transfer File SOAP/REST Connector methods (using either REST or SOAP) ; using the Online Mode adds some requirements, which are discussed in the following chapters.

4.5.4.2. Offline Mode

When sending a message using the Offline Mode through the Managed File Transfer File SOAP/REST Connector API, files must be uploaded to the MFT platform prior to calling the `sendMessage` method. This is because the operation call is not meant to include the file's body. Thus, the file(s) must be uploaded by another mean.

The Managed File Transfer File SOAP/REST Connector does not make any provision as to the method used to upload the files. A few standard possibilities include:

- FTP or FTP over SSL
- SCP, SFTP
- NFS mount
- SMB mount

A user that uses the Managed File Transfer File SOAP/REST Connector in offline mode is required to have an upload directory defined. Uploaded files must reside in this directory when a call to `sendMessage` is made. It is important that two connector clients never use the same upload directory, as it may lead to file name collisions. To ensure that an uploaded file is really the one that is being processed through a `sendMessage` call, the client may provide a SHA-256 digest of all the uploaded files. These will be checked by the MFT platform.

Important

A user upload directory is never automatically created by the MFT application. It must be created prior to using the Managed File Transfer File SOAP/REST Connector, with the following pre-requisites:

- The directory must exist as a sub-directory of the configured upload base directory (see the `connector.file.upload.basedir` property in Advanced Settings).
- The directory must be readable and writable by the `mft` UNIX user.
- The upload mechanism must be configured in such a way that the resulting uploaded file can be deleted by the `mft` UNIX user.

4.5.4.3. Online Mode

Note: this mode is only available when calling the connector using the REST interface.

When sending a message using the sendMessage operation and the REST interface, the actual files being sent can be part of the REST (HTTP) request. The files must be part of the POST request, sent as a mime multipart form. This is the same as uploading files using a browser and an HTML form. When using this method of sending a message, the following requirements apply:

- The HTTP request must be a POST request with a mime multipart form (in which data must come last).
- When authentication credentials are needed, they must be set in the request's HTTP Headers. The parameters must not be part of the multipart form but are directly part of the target POST URL (see Section 4.5.4.4, “POST Request Example”).
- the multipart HTTP form must be structured so that “simple” form parameters such as the recipient emails or the file lifetime are located before the file payload. This requirement ensures the parameter validation can be performed before the file data are sent, so that possible errors are returned as soon as possible to the caller.

Important

In Online Mode, the following limitations apply:

- The maximum total file size is limited to 2GB (sum of all file sizes in the request).

Important

When running a test call (see the `testrun` parameter), the files must not be sent alongside the parameters in the POST form.

4.5.4.4. POST Request Example

Below is the dump of a sample file upload multipart form-data; please note that the `uid` and `password` credentials are passed as HTTP Headers and NOT as part of the POST elements.

```
POST https://mft.acme.com/zephyr/connectors/REST/sendMessage?\
 OTC-Auth-Ident: YWRtaW4tbWZ0
 OTC-Auth-Password: b3BlbnRydXN0Cg==
 Content-Length: 1216
 Content-Type: multipart/form-data; boundary=---------------------------573\
438391212804753854946495 8

 ----------------------------573438391212804753854946495 8
 Content-Disposition: form-data; name=&quot;UploadType&quot;

 express
 ----------------------------573438391212804753854946495 8
 Content-Disposition: form-data; name=&quot;subject&quot;

 Very important files
 ----------------------------573438391212804753854946495 8
 Content-Disposition: form-data; name=&quot;comment&quot;

 Please review this document ASAP
 ----------------------------573438391212804753854946495 8
 Content-Disposition: form-data; name=&quot;lifetime&quot;

 10
 ----------------------------573438391212804753854946495 8
 Content-Disposition: form-data; name=&quot;encrypted&quot;

 0
 ----------------------------573438391212804753854946495 8
 Content-Disposition: form-data; name=&quot;recipients&quot;

 jsmith@another-acme.com
 ----------------------------573438391212804753854946495 8
 Content-Disposition: form-data; name=&quot;recipients&quot;
```

```
asmith@another-acme.com
----------------------------573438391212804753854946495 8
Content-Disposition: form-data; name=&quot;filename&quot;; filename=&quot;MyVeryImportantFile.doc&quot;
Content-Type: text/plain

The content of the Word Document comes here...

----------------------------573438391212804753854946495 8
Content-Disposition: form-data; name=&quot;filename&quot;; filename=&quot;MyOtherVeryImportantFile.xls&quot;
Content-Type: text/plain

The content of the Excel Document comes here...

----------------------------573438391212804753854946495 8--
```

The response sent by the server (please note the `id` attribute, that uniquely identifies the sent message):

```
Content-Type: text/plain; charset=utf-8
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: JSESSIONID=h9w9j4x82knh;Path=/zephyr
Transfer-Encoding: chunked
Server: Jetty(6.1.22)

{
    "files":     [
                {
            "index": "0",
            "download_url": "https://mft.acme.ltd/zephyr/connectors/REST/downloadFile?file=40",
            "name": "MyOtherVeryImportantFile.xls",
            "size": "13824",
            "digest": "bbea8e8a3554c1db3fb59cb37ec362bc572827ec82792de2c059615b603df3cb"
        },
                {
            "index": "1",
            "download_url": "https://mft.acme.ltd/zephyr/connectors/REST/downloadFile?file=41",
            "name": "MyVeryImportantFile.doc",
            "size": "19456"
            "digest": "bbea8e8a3554c1db3fb59cb37ec362bc572827ec82792de2c059615b603df3cb"
        }
    ],
    "recipients":     [
                {
            "index": "0",
            "email": "jsmith@another-acme.ltd"
        },
                {
            "index": "1",
            "email": "asmith@another-acme.ltd"
        }
    ],
    "subject": "Very important files",
    "type": "simple",
    "date": "20101015094523Z",
    "access_url": "https://mft.acme.ltd/zephyr/WebApp.jsp?ht=tab-MESSAGES-DETAILS-35",
    "size": "33280",
    "id": "35",
    "sender":     {
        "uid": "jdoe",
        "email": "jdoe@acme.ltd",
        "domain": "Acme"
    },
    "encrypted": "0",
    "signed": "0",
    "download_url": "https://mft.acme.ltd/zephyr/connectors/REST/downloadFile?message=35",
    "expiration_date": "20101025094523Z",
    "active": "1",
    "comment": "Please review this document ASAP"
}
```

This POST request may have been generated with the standard `curl` tool as follow:

```
curl -v -k -H "X-Otc-Auth-Uid: YWRtaW4tbWZ0" -H "X-Otc-Auth-Password: b3BlbnRydXN0Cg==" 
 -F subject="Very important files" -F lifetime=10 -F encrypted=0\
 -F comment="Please review this document ASAP" -F recipients=jsmith@another-acme.ltd\
 -F recipients=asmith@another-acme.ltd -F "file=@/home/alexis/Tmp/MyVeryImportantFile.doc"\
 -F "file=@/home/alexis/Tmp/MyOtherVeryImportantFile.xls"\
 'https://mft.acme.ltd/zephyr/connectors/REST/sendMessage'
```

Appendix A. Schema documentation for otmessage-1.3.xsd

30 november 2009

A.1. Namespace: "http://www.opentrust.com/OpenTrust/XML/Message/1.3"

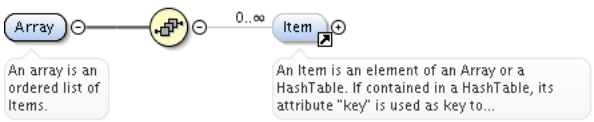
A.1.1. Schemas

A.1.1.1. Main schema **`otmessage-1.3.xsd`**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.3
Annotations	`This is the most basic type of data. This element is used to store a data value. It is encoded as a String, and its signification depends on the context.`
Properties	attribute form default: `unqualified` element form default: `qualified`

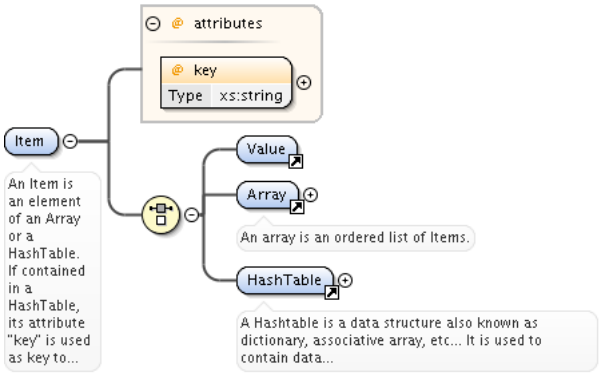
A.1.2. Elements

A.1.2.1. Element **`Array`**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.3
Annotations	`An array is an ordered list of Items.`
Diagram	
Properties	content: `complex`
Used by	Elements ErrorDetail, Header, Item, Message
Model	Item*
Children	Item

`Instance`	`<Array>`
	`  <Item key="">{0,unbounded}</Item>`
	`</Array>`

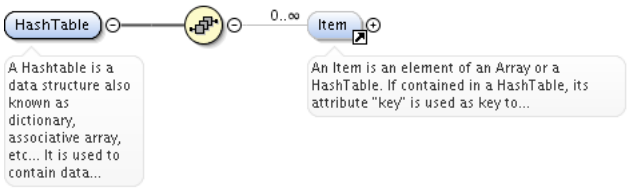
A.1.2.2. Element **`Item`**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.3
Annotations	`An Item is an element of an Array or a HashTable. If contained in a HashTable, its attribute "key" is used as key to reference the Item in the HashTable. It is not significant if the Item is contained in an Array. It is a container for one of the three "basic" data types : Value, Array and HashTable.`
Diagram	
Properties	content: `complex` nillable: `true` mixed: `false`
Used by	Elements Array, HashTable
Model	Value | Array | HashTable
Children	Array, HashTable, Value

`Instance`	`<Item key="">`
	`  <Value>{1,1}</Value>`
	`  <Array>{1,1}</Array>`
	`  <HashTable>{1,1}</HashTable>`
	`</Item>`

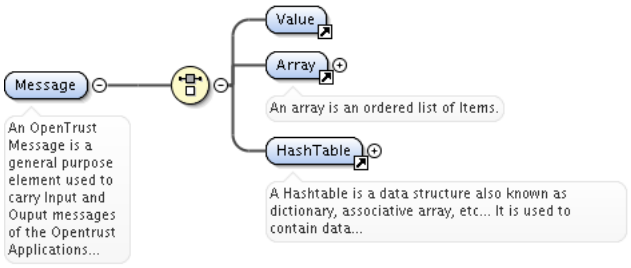
Attributes	QName: key; Type: xs:string; Fixed: ; Default: ; Use: required

A.1.2.3. Element **`HashTable`**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.3
Annotations	`A Hashtable is a data structure also known as dictionary, associative array, etc... It is used to contain data referenced by a named key. a Hashtable contains Items, and the keys are an attribute of these elements. The same key attribute MUST NOT appear in more than one Item contained in the HashTable. Applications MUST NOT rely on the order in which the containing Item appear.`
Diagram	

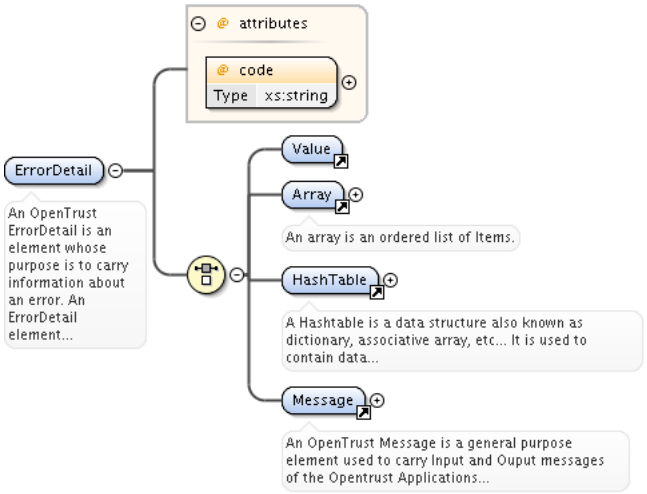
Properties	content:	`complex`
Used by	Elements	ErrorDetail, Header, Item, Message
Model	Item*	
Children	Item	
`Instance`	<HashTable> <Item key="">{0,unbounded}</Item> </HashTable>	

A.1.2.4. Element **`Message`**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.3
Annotations	An MFT Message is a general purpose element used to carry Input and Ouput messages of the Opentrust Applications SOAP connector operations. These messages are not typed, and designed to transport any structured (un-typed) data, with the help of the following containers : Array, HashTable, and Value.
Diagram	
Properties	content: `complex` mixed: `false`
Used by	Element: ErrorDetail Complex Type: OTMessageType
Model	Value | Array | HashTable
Children	Array, HashTable, Value
`Instance`	<Message> <Value>{1,1}</Value> <Array>{1,1}</Array> <HashTable>{1,1}</HashTable> </Message>

A.1.2.5. Element **`ErrorDetail`**

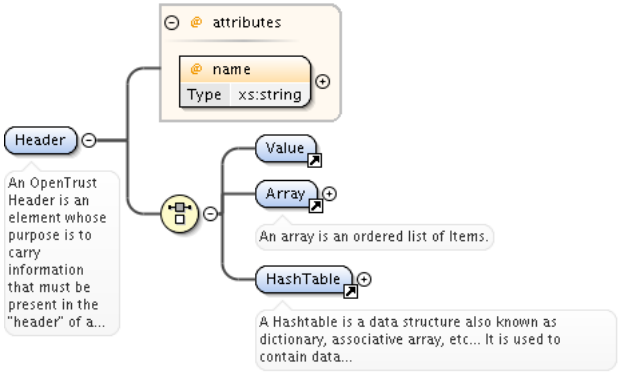
Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.3
Annotations	An OpenTrust ErrorDetail is an element whose purpose is to carry information about an error. An ErrorDetail element will always carry a "code" attribute and may contain any structured (un-typed) data, with the help of the following containers : Array, HashTable, and Value. These containers will give further information about the error. It may also contain a Message element for backward compatibility.

Diagram	
Properties	content: `complex` mixed: `false`
Used by	Complex Type: OTErrorDetailType
Model	Value | Array | HashTable | Message
Children	Array, HashTable, Message, Value

`Instance`	`<ErrorDetail code="">`
	`  <Value>{1,1}</Value>`
	`  <Array>{1,1}</Array>`
	`  <HashTable>{1,1}</HashTable>`
	`  <Message>{1,1}</Message>`
	`</ErrorDetail>`

Attributes	QName: code; Type: xs:string; Fixed: ; Default: ; Use: required

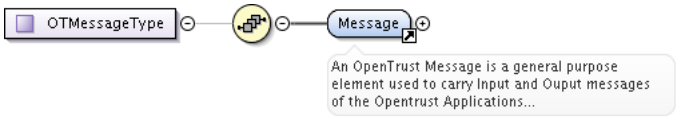
A.1.2.6. Element **Header**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.3
Annotations	An OpenTrust Header is an element whose purpose is to carry information that must be present in the "header" of a message. A Header element will always carry a "name" attribute and may contain any structured (un-typed) data, with the help of the following containers : Array, HashTable, and Value.
Diagram	

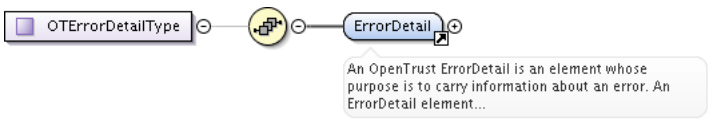
Properties	content: complex mixed: false
Used by	Complex Type: OTHeaderType
Model	Value | Array | HashTable
Children	Array, HashTable, Value
Instance	<Header name=""> <Value>{1,1}</Value> <Array>{1,1}</Array> <HashTable>{1,1}</HashTable> </Header>
Attributes	QName: name; Type: xs:string; Fixed: ; Default: ; Use: required

A.1.3. Complex Types

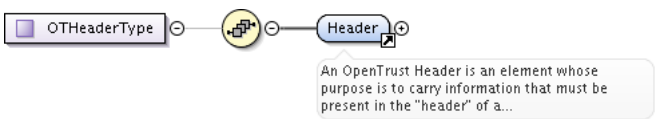
A.1.3.1. Complex Type **OTMessageType**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.3
Diagram	
Model	Message
Children	Message

A.1.3.2. Complex Type **OTErrorDetailType**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.3
Diagram	
Model	ErrorDetail
Children	ErrorDetail

A.1.3.3. Complex Type **OTHeaderType**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.3
Diagram	

Model	Header
Children	Header

A.2. Namespace: ""

A.2.1. Attributes

A.2.1.1. Attribute `Item / @key`

Namespace	No namespace
Type	xs:string
Properties	use: `required`
Used by	Element Item

A.2.1.2. Attribute `ErrorDetail / @code`

Namespace	No namespace
Type	xs:string
Properties	use: `required`
Used by	Element ErrorDetail

A.2.1.3. Attribute `Header / @name`

Namespace	No namespace
Type	xs:string
Properties	use: `required`
Used by	Element Header

Appendix B. Schema documentation for otmessage-1.4.xsd

july 4, 2012

B.1. Namespace: "http://www.opentrust.com/OpenTrust/XML/Message/1.4"

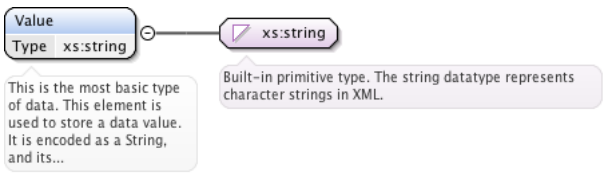
B.1.1. Schema(s)

B.1.1.1. Main schema **`otmessage-1.4.xsd`**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.4
Annotation	`This is the most basic type of data. This element is used to store a data value. It is encoded as a String, and its signification depends on the context.`
Properties	attribute form default: `unqualified` element form default: `qualified`

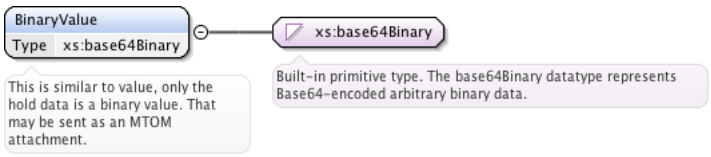
B.1.2. Element(s)

B.1.2.1. Element **`Value`**

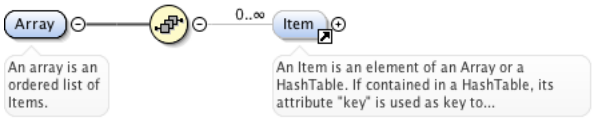
Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.4
Annotation	`This is the most basic type of data. This element is used to store a data value. It is encoded as a String, and its signification depends on the context.`
Diagram	
Type	xs:string
Properties	content: `simple` nillable: `true`

Used by	Elements	ErrorDetail, Header, Item, Message

B.1.2.2. Element **`BinaryValue`**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.4
Annotation	`This is similar to value, only the hold data is a binary value. That may be sent as an MTOM attachment.`
Diagram	
Type	xs:base64Binary
Properties	content: `simple`
Used by	Elements: Item, Message

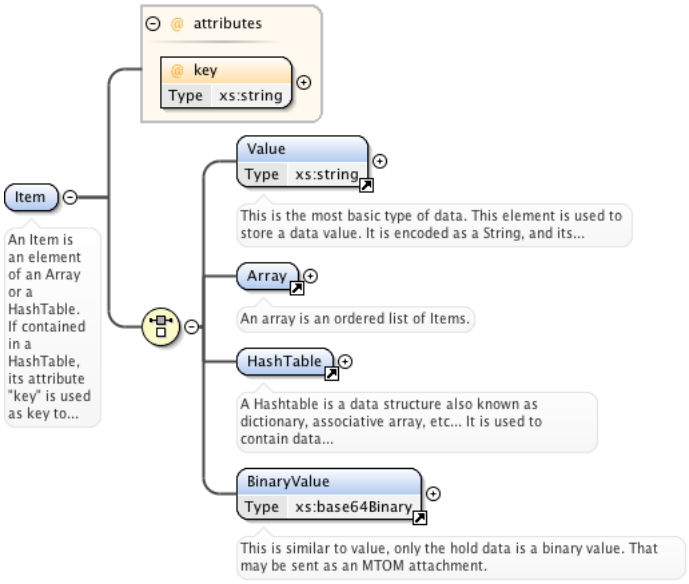
B.1.2.3. Element **`Array`**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.4
Annotation	`An array is an ordered list of Items.`
Diagram	
Properties	content: `complex`
Used by	Elements: ErrorDetail, Header, Item, Message
Model	Item*
Children	Item

Instance	`<Array xmlns="http://www.opentrust.com/OpenTrust/XML/Message/1.4">`
	`  <Item key="">{0,unbounded}</Item>`
	`</Array>`

B.1.2.4. Element **`Item`**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.4
Annotation	`An Item is an element of an Array or a HashTable. If contained in a HashTable, its attribute "key" is used as key to reference the Item in the HashTable. It is not significant if the Item is contained in an Array. It is a container for one of the three "basic" data types : Value, Array and HashTable.`

Diagram	

Properties		
	content:	`complex`
	nillable:	`true`
	mixed:	`false`

Used by		
	Elements	Array, HashTable

Model	
	Value | Array | HashTable | BinaryValue

Children	
	Array, BinaryValue, HashTable, Value

Instance

```
<Item key="" xmlns="http://www.opentrust.com/OpenTrust/XML/Message/1.4">
  <Value>{1,1}</Value>
  <Array>{1,1}</Array>
  <HashTable>{1,1}</HashTable>
  <BinaryValue>{1,1}</BinaryValue>
</Item>
```

Attributes

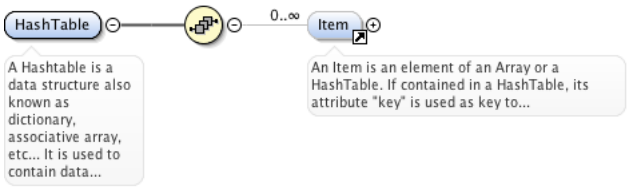
QName	Type	Fixed	Default	Use
key	xs:string			required

B.1.2.5. Element **`HashTable`**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.4

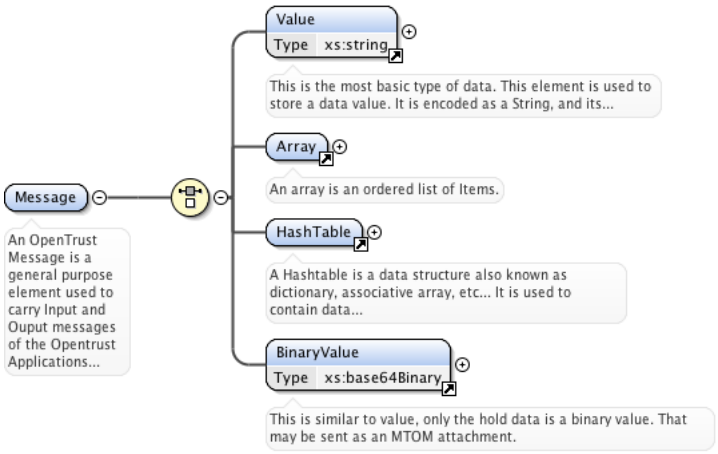
Annotation

```
A Hashtable is a data structure also
known as
dictionary, associative array, etc... It is used to contain data
referenced by a named key. a Hashtable contains Items, and the keys are
an attribute of these elements. The same key attribute MUST NOT appear
in more than one Item contained in the HashTable.
Applications MUST NOT
rely on the order in which the containing Item appear.
```

Diagram	
	

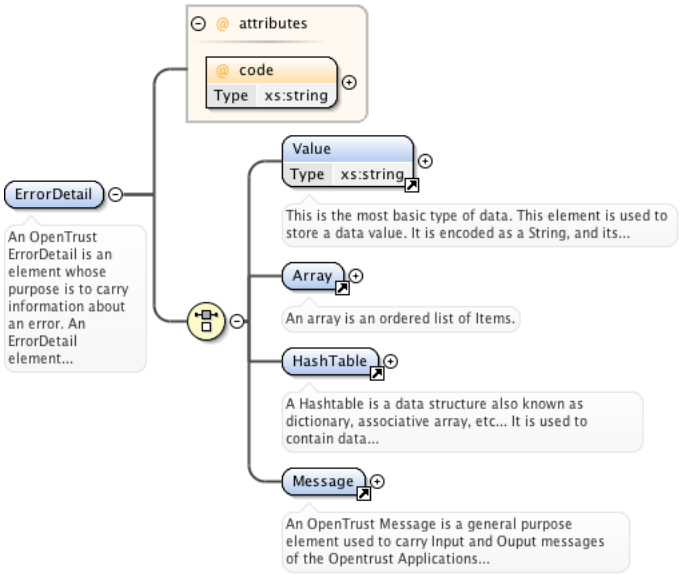
Properties	content:	complex
Used by	Elements	ErrorDetail, Header, Item, Message
Model	Item*	
Children	Item	
Instance	<HashTable xmlns="http://www.opentrust.com/OpenTrust/XML/Message/1.4"> <Item key="">{0,unbounded}</Item> </HashTable>	

B.1.2.6. Element **Message**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.4
Annotation	An MFT Message is a general purpose element used to carry Input and Ouput messages of the Opentrust Applications SOAP connector operations. These messages are not typed, and designed to transport any structured (un-typed) data, with the help of the following containers : Array, HashTable, and Value.
Diagram	
Properties	content: complex mixed: false
Used by	Element: ErrorDetail Complex Type: OTMessageType
Model	Value | Array | HashTable | BinaryValue
Children	Array, BinaryValue, HashTable, Value
Instance	<Message xmlns="http://www.opentrust.com/OpenTrust/XML/Message/1.4"> <Value>{1,1}</Value> <Array>{1,1}</Array> <HashTable>{1,1}</HashTable> <BinaryValue>{1,1}</BinaryValue> </Message>

B.1.2.7. Element **ErrorDetail**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.4

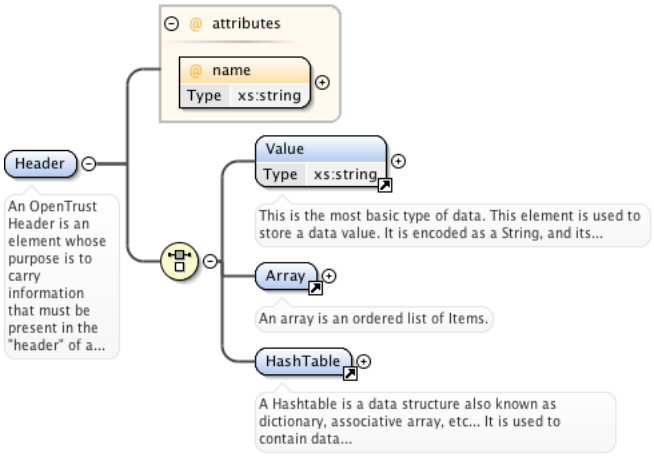
Annotation	An OpenTrust ErrorDetail is an element whose purpose is to carry information about an error. An ErrorDetail element will always carry a "code" attribute and may contain any structured (un-typed) data, with the help of the following containers : Array, HashTable, and Value. These containers will give further information about the error. It may also contain a Message element for backward compatibility.
Diagram	
Properties	content: complex mixed: false
Used by	Complex Type OTErrorDetailType
Model	Value | Array | HashTable | Message
Children	Array, HashTable, Message, Value
Instance	<ErrorDetail code="" xmlns="http://www.opentrust.com/OpenTrust/XML/Message/1.4"> <Value>{1,1}</Value> <Array>{1,1}</Array> <HashTable>{1,1}</HashTable> <Message>{1,1}</Message> </ErrorDetail>

Attributes

QName	Type	Fixed	Default	Use
code	xs:string			required

B.1.2.8. Element **Header**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.4
Annotation	An OpenTrust Header is an element whose purpose is to carry information that must be present in the "header" of a message. A Header element will always carry a "name" attribute and may contain any structured (un-typed) data, with the help of the following containers : Array, HashTable, and Value.

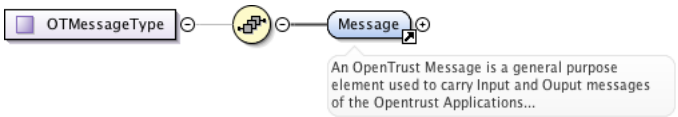
Diagram	
Properties	content: `complex` mixed: `false`
Used by	Complex Type OTHeaderType
Model	Value | Array | HashTable
Children	Array, HashTable, Value

Instance	`<Header name="" xmlns="http://www.opentrust.com/OpenTrust/XML/Message/1.4">`
	`  <Value>{1,1}</Value>`
	`  <Array>{1,1}</Array>`
	`  <HashTable>{1,1}</HashTable>`
	`</Header>`

Attributes	QName	Type	Fixed	Default	Use
	name	xs:string			required

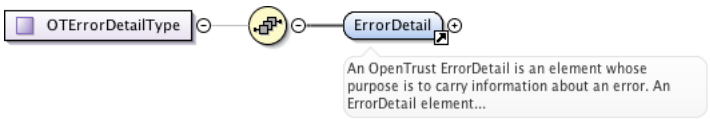
B.1.3. Complex Type(s)

B.1.3.1. Complex Type **`OTMessageType`**

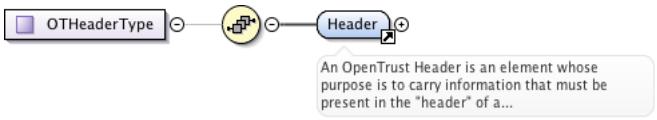
Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.4
Diagram	
Model	Message
Children	Message

B.1.3.2. Complex Type **`OTErrorDetailType`**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.4

Diagram	
Model	ErrorDetail
Children	ErrorDetail

B.1.3.3. Complex Type **`OTHeaderType`**

Namespace	http://www.opentrust.com/OpenTrust/XML/Message/1.4
Diagram	
Model	Header
Children	Header

B.2. Namespace: ""

B.2.1. Attribute(s)

B.2.1.1. Attribute **`Item / @key`**

Namespace	No namespace	
Type	xs:string	
Properties	use:	`required`
Used by	Element	Item

B.2.1.2. Attribute **`ErrorDetail / @code`**

Namespace	No namespace	
Type	xs:string	
Properties	use:	`required`
Used by	Element	ErrorDetail

B.2.1.3. Attribute **`Header / @name`**

Namespace	No namespace
Type	xs:string

Properties	use:	`required`
Used by	Element	Header